Problems and Solutions for Android Developers

Android Cookbook

Free Sampler

O'REILLY®

Edited by Ian F. Darwin

Android Cookbook

Jump in and build working Android apps with the help of more than 200 tested recipes. With this cookbook, you'll find solutions for working with the user interfaces, multitouch gestures, location awareness, web services, and device features such as the phone, camera, and accelerometer. You also get useful steps on packaging your app for the Android Market.

Ideal for developers familiar with Java, Android basics, and the Java SE API, this book features recipes contributed by more than three dozen developers from the Android community. Each recipe provides a clear solution and sample code you can use in your project right away. Among numerous topics, this cookbook helps you:

- Use guidelines for designing a successful Android app
- Work with UI controls, effective layouts, and graphical elements
- Learn how to take advantage of Android's rich features in your app
- Save and retrieve application data in files, SD cards, and embedded databases
- Access RESTful web services, RSS/Atom feeds, and information from websites
- Create location-aware services to find locations and landmarks, and situate them on Google Maps and OpenStreetMap
- Test and troubleshoot individual components and your entire application

Ian F. Darwin is a software developer, consultant, and teacher with more than 30 years of experience in the computer industry. He's the author of *Checking C Programs with Lint* and *Java Cookbook* (both O'Reilly), as well as more than a hundred articles and courses on technologies such as Unix, C, Java, Mac, Python, Android, and the Web.

"This book is here to help the Android developer community share the knowledge . . . Those who contribute knowledge here are helping to make Android development easier for those who come after."

—from the Preface

US $54.99 CAN $57.99
ISBN: 978-1-449-38841-6

Twitter: @oreillymedia
facebook.com/oreilly

O'REILLY®
oreilly.com

Want to read more?

You can buy this book at **oreilly.com**
in print and ebook format.

Buy 2 books, get the 3rd FREE!
Use discount code: OPC10
All orders over $29.95 qualify for **free shipping** within the US.

It's also available at your favorite book retailer,
including the iBookstore, the Android Marketplace,
and Amazon.com.

O'REILLY®

Spreading the knowledge of innovators

oreilly.com

Android Cookbook
by Ian F. Darwin

Copyright © 2012 O'Reilly Media, Inc.. All rights reserved.
Printed in the United States of America.

Published by O'Reilly Media, Inc, 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://my.safaribooksonline.com*). For more information, contact our corporate/institutional sales department: (800) 998-9938 or *corporate@oreilly.com*.

Editors: Mike Loukides and Courtney Nash
Production Editor: Teresa Elsey
Copyeditor: Audrey Doyle
Proofreader: Stacie Arellano

Indexer: Lucie Haskins
Cover Designer: Karen Montgomery
Interior Designer: David Futato
Illustrators: Robert Romano and Rebecca Demarest

April 2012: First Edition.

Revision History for the First Edition:
2012-04-05 First release
See *http://oreilly.com/catalog/errata.csp?isbn=9781449388416* for release details.

Nutshell Handbook, the Nutshell Handbook logo, and the O'Reilly logo are registered trademarks of O'Reilly Media, Inc. *Android Cookbook*, the image of a marine iguana, and related trade dress are trademarks of O'Reilly Media, Inc.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and O'Reilly Media, Inc., was aware of a trademark claim, the designations have been printed in caps or initial caps.

While every precaution has been taken in the preparation of this book, the publisher and authors assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

ISBN: 978-1-449-38841-6

[LSI]

1333645094

Table of Contents

Preface ..... **xiii**

1. Getting Started **1**
1.1 Introduction: Getting Started 1
1.2 Learning the Java Language 1
1.3 Creating a "Hello, World" Application from the Command Line 3
1.4 Creating a "Hello, World" Application in Eclipse 6
1.5 Setting Up an IDE on Windows to Develop for Android 13
1.6 Understanding the Android Life Cycle 20
1.7 Installing .apk Files onto an Emulator via the ADB 21
1.8 Installing Apps onto an Emulator via SlideME 22
1.9 Sharing Java Classes from Another Eclipse Project 23
1.10 Referencing Libraries to Implement External Functionality 26
1.11 Using SDK Samples to Help Avoid Head Scratching 29
1.12 Keeping the Android SDK Updated 32
1.13 Taking a Screenshot from the Emulator/Android Device 39
1.14 Program: A Simple CountDownTimer Example 41
1.15 Program: Tipster, a Tip Calculator for the Android OS 44

2. Designing a Successful Application **63**
2.1 Introduction: Designing a Successful Android Application 63
2.2 Exception Handling 66
2.3 Accessing Android's Application Object as a "Singleton" 69
2.4 Keeping Data When the User Rotates the Device 71
2.5 Monitoring the Battery Level of an Android Device 74
2.6 Creating Splash Screens in Android 75
2.7 Designing a Conference/Camp/Hackathon/Institution App 79
2.8 Using Google Analytics in an Android Application 81
2.9 A Simple Torch/Flashlight 83
2.10 Adapting an Android Phone Application to Be Used on a Tablet 86
2.11 Setting First-Run Preferences 88

2.12 Formatting the Time and Date for Display 89
2.13 Controlling Input with KeyListeners 91
2.14 Backing Up Android Application Data 95
2.15 Using Hints Instead of Tool Tips 101

3. Testing 103
3.1 Introduction: Testing 103
3.2 Doing Test-Driven Development (TDD) in Android 103
3.3 Setting Up an Android Virtual Device (AVD) for App Testing 104
3.4 Testing on a Huge Range of Devices with Cloud-based Testing 113
3.5 Creating and Using a Test Project 114
3.6 Troubleshooting Application Crashes 118
3.7 Debugging Using Log.d and LogCat 122
3.8 Getting Bug Reports from Users Automatically with BugSense 123
3.9 Using a Local Runtime Application Log for Analysis of Field Errors or Situations 125
3.10 Reproducing Activity Life-Cycle Scenarios for Testing 129
3.11 Keeping Your App Snappy with StrictMode 134
3.12 Running the Monkey Program 135
3.13 Sending Text Messages and Placing Calls Between AVDs 137

4. Inter-/Intra-Process Communication 141
4.1 Introduction: Inter-/Intra-Process Communication 141
4.2 Opening a Web Page, Phone Number, or Anything Else with an Intent 142
4.3 Emailing Text from a View 143
4.4 Sending an Email with Attachments 146
4.5 Pushing String Values Using Intent.putExtra() 147
4.6 Retrieving Data from a Subactivity Back to Your Main Activity 149
4.7 Keeping a Service Running While Other Apps Are on Display 151
4.8 Sending/Receiving a Broadcast Message 153
4.9 Starting a Service After Device Reboot 154
4.10 Creating a Responsive Application Using Threads 155
4.11 Using AsyncTask to Do Background Processing 157
4.12 Sending Messages Between Threads Using an Activity Thread Queue and Handler 165
4.13 Creating an Android Epoch HTML/JavaScript Calendar 167

5. Content Providers 173
5.1 Introduction: Content Providers 173
5.2 Retrieving Data from a Content Provider 173
5.3 Writing a Content Provider 175
5.4 Writing an Android Remote Service 177

6. Graphics 183
6.1 Introduction: Graphics 183
6.2 Using a Custom Font 183
6.3 Drawing a Spinning Cube with OpenGL ES 186
6.4 Adding Controls to the OpenGL Spinning Cube 190
6.5 Freehand Drawing Smooth Curves 193
6.6 Taking a Picture Using an Intent 199
6.7 Taking a Picture Using android.media.Camera 201
6.8 Scanning a Barcode or QR Code with the Google ZXing Barcode Scanner 204
6.9 Using AndroidPlot to Display Charts and Graphs 208
6.10 Using Inkscape to Create an Android Launcher Icon 210
6.11 Creating Easy Launcher Icons from OpenClipArt.org Using Paint.NET 217
6.12 Using Nine Patch Files 224
6.13 Creating HTML5 Charts with Android RGraph 227
6.14 Adding a Simple Raster Animation 231
6.15 Using Pinch to Zoom 234

7. Graphical User Interface 239
7.1 Introduction: GUI 239
7.2 Understanding and Following User Interface Guidelines 240
7.3 Handling Configuration Changes by Decoupling the View from the Model 241
7.4 Creating a Button and Its Click Event Listener 244
7.5 Wiring Up an Event Listener in Five Different Ways 245
7.6 Using CheckBoxes and RadioButtons 250
7.7 Enhancing UI Design Using Image Buttons 253
7.8 Offering a Drop-Down Chooser via the Spinner Class 256
7.9 Handling Long-Press/Long-Click Events 258
7.10 Displaying Text Fields with TextView and EditText 259
7.11 Constraining EditText Values with Attributes and the TextWatcher Interface 260
7.12 Implementing AutoCompleteTextView 263
7.13 Feeding AutoCompleteTextView Using an SQLite Database Query 265
7.14 Turning Edit Fields into Password Fields 267
7.15 Changing the Enter Key to “Next” on the Soft Keyboard 268
7.16 Processing Key-Press Events in an Activity 270
7.17 Let Them See Stars: Using RatingBar 272
7.18 Making a View Shake 276
7.19 Providing Haptic Feedback 277
7.20 Navigating Different Activities Within a TabView 281
7.21 Creating a Custom Title Bar 283

7.22 Formatting Numbers 285
7.23 Formatting with Correct Plurals 289
7.24 Starting a Second Screen from the First 292
7.25 Creating a Loading Screen That Will Appear Between Two Activities 301
7.26 Using SlidingDrawer to Overlap Other Components 303
7.27 Customizing the SlidingDrawer Component to Animate/Transition from the Top Down 305
7.28 Adding a Border with Rounded Corners to a Layout 307
7.29 Detecting Gestures in Android 309
7.30 Building a UI Using Android 3.0 Fragments in Android 1.6 and Later 316
7.31 Using the Android 3.0 Photo Gallery 321
7.32 Creating a Simple App Widget 324

8. GUI Alerts: Menus, Dialogs, Toasts, and Notifications 329
8.1 Introduction: GUI Alerts 329
8.2 Creating and Displaying a Menu 330
8.3 Handling Choice Selection in a Menu 331
8.4 Creating a Submenu 333
8.5 Creating a Pop-up/Alert Dialog 336
8.6 Using a Timepicker Widget 338
8.7 Creating an iPhone-like Wheel Picker for Selection 340
8.8 Creating a Tabbed Dialog 343
8.9 Creating a ProgressDialog 346
8.10 Creating a Custom Dialog with Buttons, Images, and Text 347
8.11 Creating a Reusable About Box Class 349
8.12 Customizing the Appearance of a Toast 353
8.13 Creating a Notification in the Status Bar 354

9. GUI: ListView 361
9.1 Introduction: ListView 361
9.2 Building List-Based Applications with ListView 361
9.3 Creating a "No Data" View for ListViews 366
9.4 Creating an Advanced ListView with Images and Text 367
9.5 Using Section Headers in ListViews 372
9.6 Keeping the ListView with the User's Focus 376
9.7 Writing a Custom List Adapter 377
9.8 Handling Orientation Changes: From ListView Data Values to Landscape Charting 381

10. Multimedia 387
10.1 Introduction: Multimedia 387

10.2 Playing a YouTube Video 387
10.3 Using the Gallery with the ImageSwitcher View 388
10.4 Capturing Video Using MediaRecorder 391
10.5 Using Android's Face Detection Capability 394
10.6 Playing Audio from a File 398
10.7 Playing Audio Without Interaction 400
10.8 Using Speech to Text 402
10.9 Making the Device Speak with Text-to-Speech 403

11. Data Persistence 407
11.1 Introduction: Data Persistence 407
11.2 Getting File Information 407
11.3 Reading a File Shipped with the App Rather Than in the Filesystem 411
11.4 Listing a Directory 413
11.5 Getting Total and Free Space Information About the SD Card 414
11.6 Providing User Preference Activity with Minimal Effort 415
11.7 Checking the Consistency of Default Shared Preferences 419
11.8 Performing Advanced Text Searches 421
11.9 Creating an SQLite Database in an Android Application 427
11.10 Inserting Values into an SQLite Database 428
11.11 Loading Values from an Existing SQLite Database 428
11.12 Working with Dates in SQLite 429
11.13 Parsing JSON Using JSONObject 432
11.14 Parsing an XML Document Using the DOM API 433
11.15 Parsing an XML Document Using an XmlPullParser 435
11.16 Adding a Contact 439
11.17 Reading Contact Data 442

12. Telephone Applications 445
12.1 Introduction: Telephone Applications 445
12.2 Doing Something When the Phone Rings 445
12.3 Processing Outgoing Phone Calls 449
12.4 Dialing the Phone 453
12.5 Sending Single-Part or Multipart SMS Messages 454
12.6 Receiving an SMS Message in an Android Application 457
12.7 Using Emulator Controls to Send SMS Messages to the Emulator 458
12.8 Using Android's TelephonyManager to Obtain Device Information 459

13. Networked Applications 471
13.1 Introduction: Networking 471
13.2 Using a RESTful Web Service 472
13.3 Extracting Information from Unstructured Text Using Regular Expressions 474

13.4 Parsing RSS/Atom Feeds Using ROME 476
13.5 Using MD5 to Digest Clear Text 481
13.6 Converting Text into Hyperlinks 481
13.7 Accessing a Web Page Using WebView 482
13.8 Customizing a WebView 484

14. Gaming and Animation 485
14.1 Introduction: Gaming and Animation 485
14.2 Building an Android Game Using flixel-android 486
14.3 Building an Android Game Using AndEngine (Android-Engine) 489
14.4 Processing Timed Keyboard Input 495

15. Social Networking 497
15.1 Introduction: Social Networking 497
15.2 Integrating Social Networking Using HTTP 497
15.3 Loading a User's Twitter Timeline Using JSON 500

16. Location and Map Applications 503
16.1 Introduction: Location-Aware Applications 503
16.2 Getting Location Information 503
16.3 Accessing GPS Information in Your Application 505
16.4 Mocking GPS Coordinates on a Device 508
16.5 Using Geocoding and Reverse Geocoding 510
16.6 Getting Ready for Google Maps Development 511
16.7 Adding a Device's Current Location to Google Maps 517
16.8 Drawing a Location Marker on a Google MapView 519
16.9 Drawing Multiple Location Markers on a MapView 523
16.10 Creating Overlays for a Google MapView 528
16.11 Changing Modes of a Google MapView 529
16.12 Drawing an Overlay Icon Without Using a Drawable 530
16.13 Implementing Location Search on Google Maps 535
16.14 Placing a MapView Inside a TabView 537
16.15 Handling a Long-Press in a MapView 541
16.16 Using OpenStreetMap 544
16.17 Creating Overlays in OpenStreetMap Maps 547
16.18 Using a Scale on an OpenStreetMap Map 550
16.19 Handling Touch Events on an OpenStreetMap Overlay 551
16.20 Getting Location Updates with OpenStreetMap Maps 554

17. Accelerometer 559
17.1 Introduction: Sensors 559
17.2 Checking for the Presence or Absence of a Sensor 560
17.3 Using the Accelerometer to Detect Shaking of the Device 561

17.4 Checking Whether a Device Is Facing Up or Facing Down Based on Screen Orientation Using an Accelerometer 564
17.5 Finding the Orientation of an Android Device Using an Orientation Sensor 565
17.6 Reading the Temperature Sensor 567

18. Bluetooth 569
18.1 Introduction: Bluetooth 569
18.2 Enabling Bluetooth and Making the Device Discoverable 569
18.3 Connecting to a Bluetooth-Enabled Device 571
18.4 Listening for and Accepting Bluetooth Connection Requests 574
18.5 Implementing Bluetooth Device Discovery 575

19. System and Device Control 577
19.1 Introduction: System and Device Control 577
19.2 Accessing Phone Network/Connectivity Information 577
19.3 Obtaining Information from the Manifest File 578
19.4 Changing Incoming Call Notification to Silent, Vibrate, or Normal 579
19.5 Copying Text and Getting Text from the Clipboard 581
19.6 Using LED-Based Notifications 582
19.7 Making the Device Vibrate 583
19.8 Running Shell Commands from Your Application 584
19.9 Determining Whether a Given Application Is Running 586

20. Other Programming Languages and Frameworks 587
20.1 Introduction: Other Programming Languages 587
20.2 Running an External/Native Unix/Linux Command 588
20.3 Running Native C/C++ Code with JNI on the NDK 589
20.4 Getting Started with the Scripting Layer for Android (SL4A, Formerly Android Scripting Environment) 594
20.5 Creating Alerts in SL4A 597
20.6 Fetching Your Google Documents and Displaying Them in a ListView Using SL4A 600
20.7 Sharing SL4A Scripts in QR Codes 603
20.8 Using Native Handset Functionality from WebView via JavaScript 607
20.9 Creating a Platform-Independent Application Using PhoneGap/Cordova 608

21. Strings and Internationalization 611
21.1 Introduction: Internationalization 611
21.2 Internationalizing Application Text 612
21.3 Finding and Translating Strings 615

21.4 Handling the Nuances of strings.xml 617

22. Packaging, Deploying, and Distributing/Selling Your App 623
22.1 Introduction: Packaging, Deploying, and Distributing 623
22.2 Creating a Signing Certificate 623
22.3 Signing Your Application 626
22.4 Distributing Your Application via Android Play (formerly the Android Market) 627
22.5 Integrating AdMob into Your App 629
22.6 Obfuscating and Optimizing with ProGuard 633
22.7 Providing a Link to Other Published Apps in the Google Play Market 636

Index 641

CHAPTER 1

Getting Started

1.1 Introduction: Getting Started

Ian Darwin

Discussion

The famous "Hello, World" pattern came about when Kernighan and Plaugher wanted to write a "recipe" on how to get started in any new programming language and environment. This chapter is affectionately dedicated to these fine gentlemen, and to everyone who has ever struggled to get started in a new programming paradigm.

1.2 Learning the Java Language

Ian Darwin

Problem

Android apps are written in the Java programming language before they are converted into Android's own class file format, DEX. If you don't know how to program in Java you will find it hard to write Android apps.

Solution

Lots of resources are available for learning Java. Most of them will teach you what you need, but will also mention some API classes that are not available for Android development. *Avoid* any sections in any resource that talk about topics listed in the lefthand column of Table 1-1.

Table 1-1. Parts of the Java API to ignore

Java API	Android equivalent
Swing, applets	Android's GUI; see Chapter 7.
Application entry point `main()`	See Recipe 1.6.
J2ME/Java ME	Most of `android.*` replaces Java ME API.
Servlets/JSP, J2EE/Java EE	Designed for server-side use.

Discussion

Here are some books and resources on Java programming:

- *Java in a Nutshell* by David Flanagan (O'Reilly) is a good introduction for programmers, particularly those who are coming from C/C++. This book has grown from an acorn to a coconut in size, to keep up with the growth of Java SE over its lifetime.
- *Head First Java* by Kathy Sierra and Bert Bates (O'Reilly). This provides a great visual-learner-oriented introduction to the language.
- *Thinking in Java* by Bruce Eckel (Prentice-Hall).
- *Learning Java* by Patrick Niemeyer and Jonathan Knudsen (O'Reilly).
- "Great Java: Level 1", a video by Brett McLaughlin (O'Reilly). This provides a visual introduction to the language.
- *Java: The Good Parts* by Jim Waldo (O'Reilly).
- *Java Cookbook*, which I wrote and which O'Reilly published. This is regarded as a good second book for Java developers. It has entire chapters on strings, regular expressions, numbers, dates and time, structuring data, I/O and directories, internationalization, threading, and networking, all of which apply to Android. It also has a number of chapters that are specific to Swing and to some EE-based technologies.

Please understand that this list will probably never be completely up-to-date. You should also refer to O'Reilly's freely downloadable (with registration) *Android Development Bibliography*, a compilation of all the books from the various publishers whose books are in the online Safari service. This book is also distributed without charge at relevant conferences where O'Reilly has a booth.

See Also

This book's primary author maintains a list of Java resources online at *http://www.darwinsys.com/java/*.

O'Reilly has many of the best Java books around; there's a complete list at *http://oreilly.com/pub/topic/java*.

1.3 Creating a "Hello, World" Application from the Command Line

Ian Darwin

Problem

You want to create a new Android project without using the Eclipse ADT plug-in.

Solution

Use the Android Development Kit (ADK) tool `android` with the `create project` argument and some additional arguments to configure your project.

Discussion

In addition to being the name of the platform, `android` is also the name of a command-line tool for creating, updating, and managing projects. You can either navigate into the *android-sdk-xxx* directory, or you can set your `PATH` variable to include its *tools* subdirectory.

Then, to create a new project, give the command `android create project` with some arguments. Example 1-1 is an example run under MS-DOS.

Example 1-1. Creating a new project

```
C:> PATH=%PATH%;"C:\Documents and Settings\Ian\My Documents\android-sdk-windows\tools"; \
          "C:\Documents and Settings\Ian\My Documents\android-sdk-windows\platform-tools"
C:> android create project --target android-7 --package com.example.foo
     --name Foo --activity FooActivity --path .\MyAndroid
Created project directory: C:\Documents and Settings\Ian\My Documents\MyAndroid
Created directory C:\Documents and Settings\Ian\My Documents\MyAndroid\src\com\example\foo
Added file C:\Documents and Settings\Ian\My
    Documents\MyAndroid\src\com\example\foo\FooActivity.java
Created directory C:\Documents and Settings\Ian\My Documents\MyAndroid\res
Created directory C:\Documents and Settings\Ian\My Documents\MyAndroid\bin
Created directory C:\Documents and Settings\Ian\My Documents\MyAndroid\libs
Created directory C:\Documents and Settings\Ian\My Documents\MyAndroid\res\values
Added file C:\Documents and Settings\Ian\My Documents\MyAndroid\res\values\strings.xml
Created directory C:\Documents and Settings\Ian\My Documents\MyAndroid\res\layout
Added file C:\Documents and Settings\Ian\My Documents\MyAndroid\res\layout\main.xml
Added file C:\Documents and Settings\Ian\My Documents\MyAndroid\AndroidManifest.xml
Added file C:\Documents and Settings\Ian\My Documents\MyAndroid\build.xml

C:>
```

Table 1-2 lists the arguments for the `create project` code.

Table 1-2. List of create project arguments

Name	Meaning	Example
`--activity`	Name of your "main class" and default name for the generated *.apk* file.	`--activity HelloActivity`
`--name`	Name of the project and the generated *.apk* file.	`--name MyProject`
`--package`	Name of the Java package for your classes.	`--package com.example.hello`
`--path`	Path to create the project in (does not create a subdirectory under this, so don't use `/home/`*you*`/workspace`, but rather `/home/`*you*`/workspace/`*NewProjectName*).	`--path /home/ian/workspace/MyProject` (see above for Windows example)
`--target`	API level of the Android platform to target; use `android list targets` to see list of targets. A number is an "ID," not an API level; for that, use `android-` with the API level you want.	`--target android-10`

If it cannot complete the requested operation, the `android` command presents a voluminous "command usage" message listing all the operations it can do and the arguments for them. If successful, the `android create project` command creates the files and directories listed in Table 1-3.

Table 1-3. Artifacts created by create project

Name	Meaning
AndroidManifest.xml	Config file that tells Android about your project
bin	Generated binaries (compiled class files)
build.properties	Editable properties file
build.xml	Standard Ant build control file
default.properties or *project.properties* (depending on tools version)	Stores SDK version and libraries used; maintained by plug-in
gen	Generated stuff
libs	Libraries, of course
res	Important resource files (*strings.xml*, layouts, etc.)
src	Source code for your application
src/packagename/ActivityName.java	Source of "main" starting activity
test	Copies of most of the above

It is a normal and recommended Android practice to create your user interface in XML using the layout file created under `res/layout`, but it is certainly possible to write all the code in Java. To keep this example self-contained, we'll do it the "wrong" way for now. Use your favorite text editor to replace the contents of the file *HelloWorld.java* with the contents of Example 1-2.

Example 1-2. HelloWorld.java

```
import android.app.Activity;
import android.widget.*;

public class Hello extends Activity {

        /**
         * This method gets invoked when the activity is instantiated in
         * response to e.g., you clicked on the app's Icon in the Home Screen.
         */
        @Override
        public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                // Create a TextView for the current Activity
                TextView view = new TextView(this);
                // Make it say something
                view.setText("Hello World");
                // Put this newly created view into the Activity,
                // sort of like JFrame.getContentPane().add(view)
                setContentView(view);
        }
}
```

Assuming you have the Apache Software Foundation Ant Build Tool installed (and it is included with recent versions of the Android SDK), you can now (in a command-line window) change to the project directory (`...MyDocuments\MyAndroid` in Example 1-1) and issue the command:

```
ant debug
```

This will create an archive file named, for example, *MyAndroid.apk* (with "apk" standing for Android Package) in the ***`bin`*** directory.

If this is your first time here, you may need to create an Android Virtual Device (AVD), which is just a named configuration for the Android emulator specifying target resolution, API level, and so on. You can create an emulator using:

```
android create avd -n my_droid -t 7
```

For more details on creating an AVD, see Recipe 3.3.

You can then start the Android Debug Bridge (ADB) server and the emulator:

```
adb start-server
emulator -avd my_droid -t 5
```

Assuming you now have either the emulator running or your device plugged in and recognized via USB, you can then do:

```
adb -e install -r bin/MyAndroid.apk
```

The `-e` flag is for the emulator; use `-d` for a real device.

If you are handy with shell scripts or batch files, you'll want to create one called, say, *download*, to avoid typing the `adb` invocation on every build cycle.

Finally you can start your app! You can use the Application list: tap the little icon that looks like a 5×5 row of dots, scroll to your application by name, and tap its icon.

You will probably find it convenient to create an icon for your app on the home screen of the device or emulator; this icon will survive multiple `install -r` cycles, so it's the easiest way to test the running of your application.

See Also

Recipe 1.4. The blog "a little madness" has a more detailed formulation. The official Android reference site has a page on developing without Eclipse.

1.4 Creating a "Hello, World" Application in Eclipse

Ian Darwin

Problem

You want to use Eclipse to develop your Android application.

Solution

Install Eclipse, the Android SDK, and the ADT plug-in. Create your project and start writing your app. Build it, and test it under the emulator, from within Eclipse.

Discussion

Once you have these items installed, you are ready to begin:

- Eclipse IDE
- The Android SDK
- The ADT plug-in

If you want a more detailed exposition of installing these three items, please refer to Recipe 1.5.

To get started, create a new project from the File→New menu (see Figure 1-1).

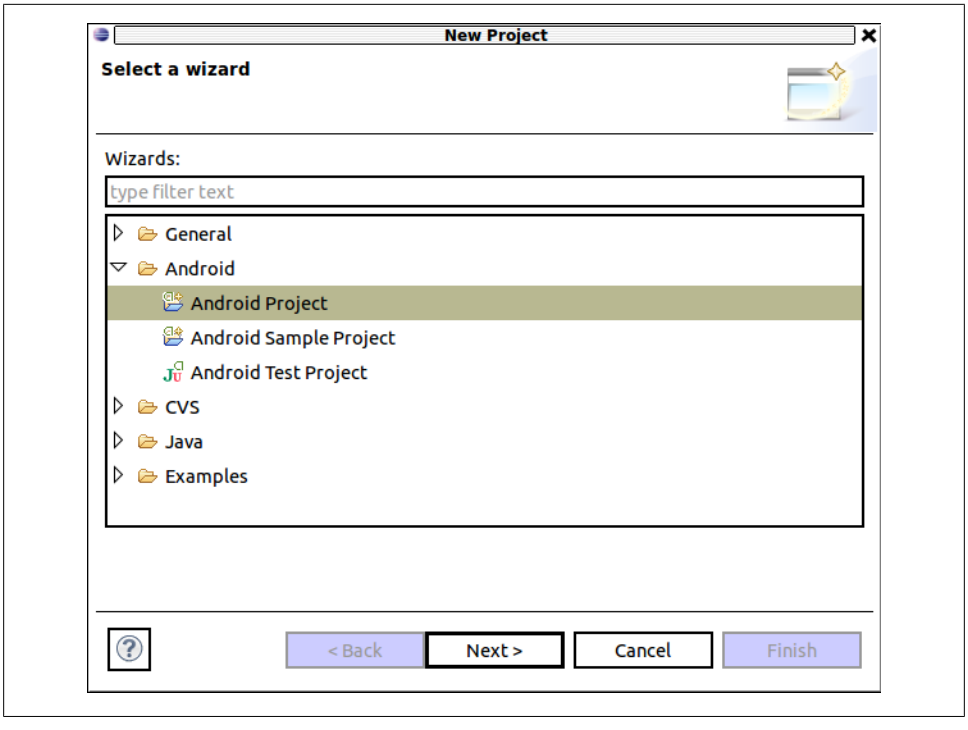


Figure 1-1. Starting to create an Eclipse project

Click Next. Give your new project a name, and click Next (see Figure 1-2).

Select an SDK version to target. Version 2.1 gives you almost all the devices in use today; version 3.x or 4.x gives you the latest features (see Figure 1-3). You decide.

Figure 1-4 shows the project structure expanded in the Project panel on the right. It also shows the extent to which you can use Eclipse auto-completion within Android—I added the `gravity` attribute for the label, and Eclipse is offering a full list of possible attribute values. I chose `center-horizontal`, so the label should be centered when we get the application running.

In fact, if you set `gravity` to `center_vertical` on the `LinearLayout` *and* set it to `center_horizontal` on the `TextView`, the text will be centered both vertically and horizontally. Example 1-3 is the layout file *main.xml* (located under *res/layout*) which achieves this.

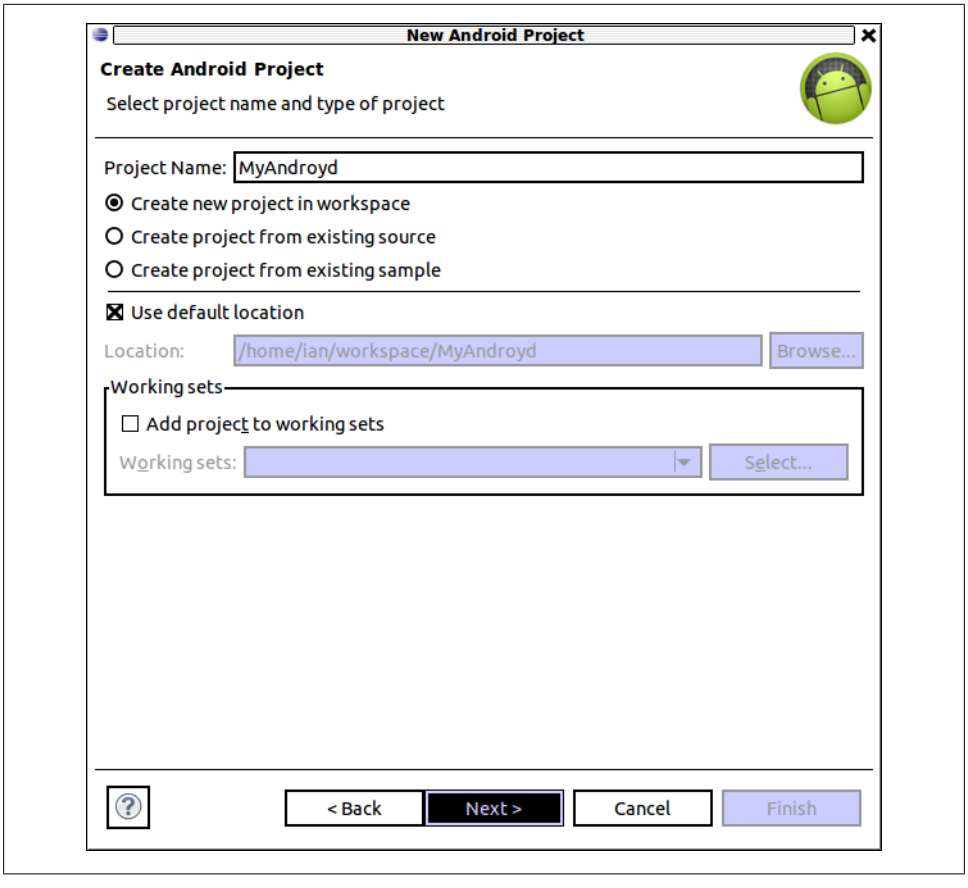


Figure 1-2. Setting parameters for a new Eclipse project

Example 1-3. The XML layout

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:gravity="center_vertical"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello"
    android:gravity="center_horizontal"
    />
</LinearLayout>
```

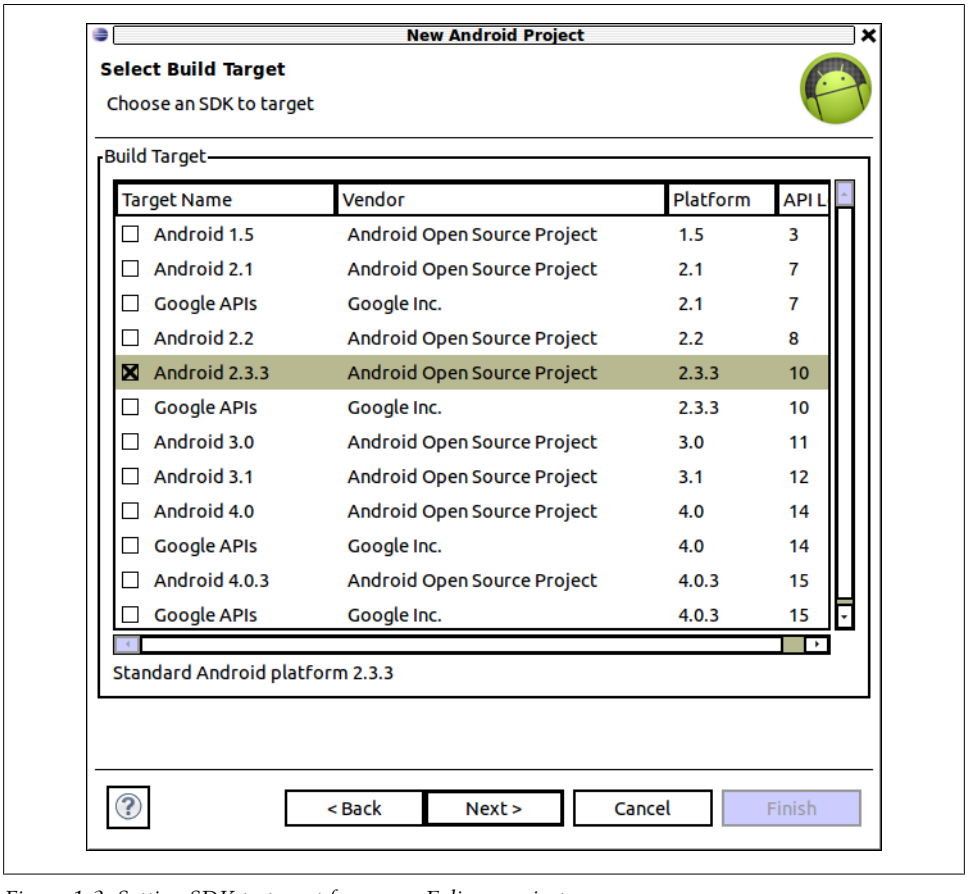


Figure 1-3. Setting SDK to target for a new Eclipse project

As always, Eclipse generates a compiled version whenever you save a source file. Also, in an Android project, it also runs an Ant build to create the compiled, packaged APK that is ready to run. So you only need to run it. Right-click on the project itself and select Run As → Android Project. (See Figure 1-5.)

This will start the Android emulator if it's not already running. The emulator will start with the word *Android* in typewriter text, then switch to the fancier Android font with a moving white patch over blue lettering—remember the Microsoft Windows 95 start-up? See Figure 1-6.

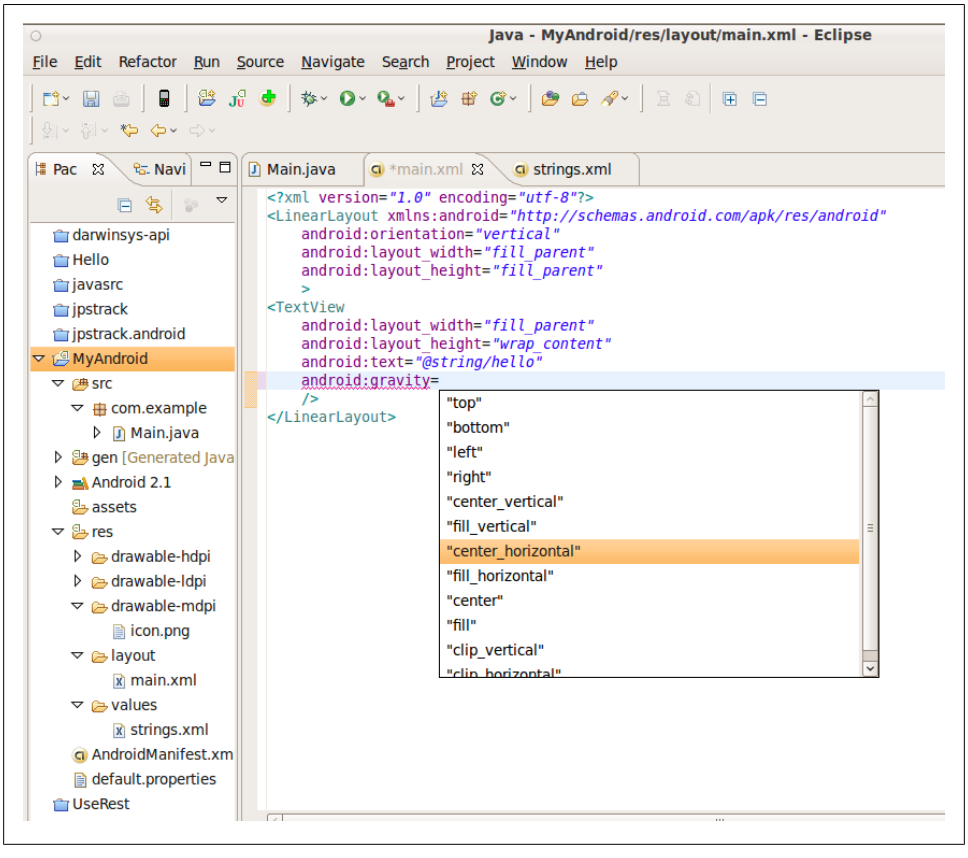


Figure 1-4. Using the Eclipse editor to set gravity on a TextView

After a little longer, your application should start up (Figure 1-5 only shows a screenshot of the application itself, since the rest of the emulator view is redundant). See Figure 1-7.

See Also

Recipe 1.3

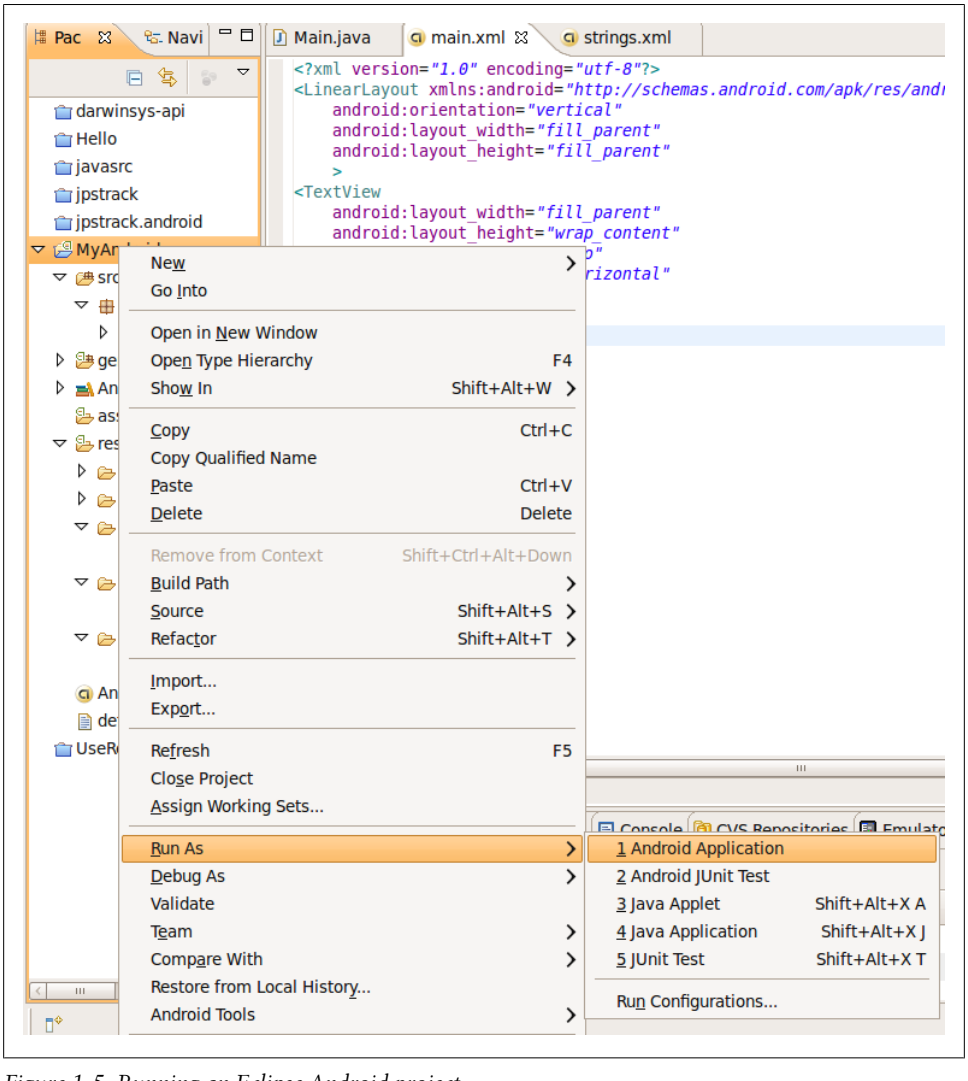


Figure 1-5. Running an Eclipse Android project

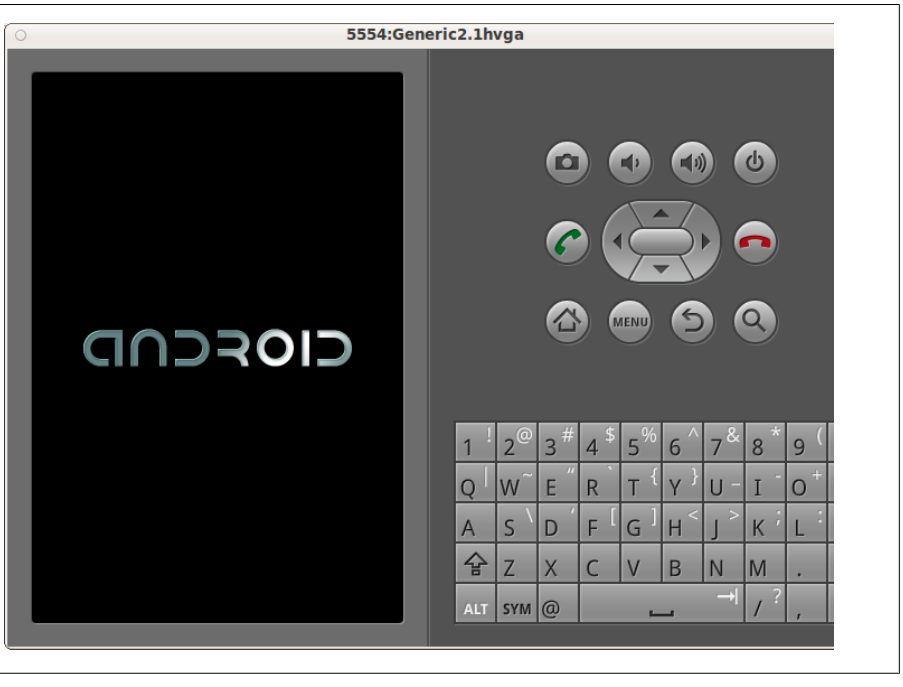


Figure 1-6. The Android project starting up in the emulator

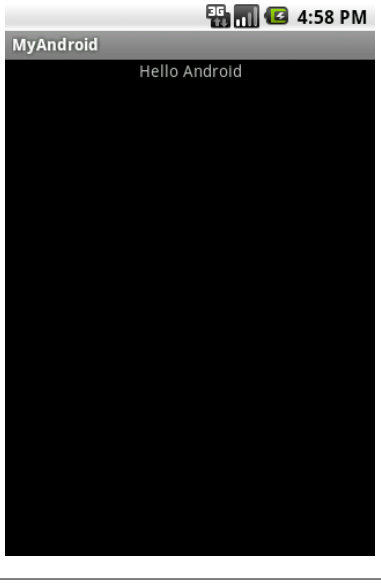


Figure 1-7. The Eclipse project running in the emulator

1.5 Setting Up an IDE on Windows to Develop for Android

Daniel Fowler

Problem

You want to develop your Android applications using a Windows PC, so a concise guide to setting up an IDE for that platform is useful.

Solution

The use of the Eclipse IDE is recommended when developing Android apps. Configuring Eclipse on Windows is not a single-shot install; several stages need to be completed. This recipe provides details on those stages.

Discussion

To develop applications for Android, the Eclipse Integrated Development Environment (IDE) for Java is recommended. An Android Development Tools (ADT) plug-in is available to enhance Eclipse. The ADT plug-in uses the Android Software Development Kit (SDK) which provides essential programs for developing Android software. To set up a development system you will need to download and install the following:

- Java Standard Edition Development Kit
- Eclipse for Java Development
- Android Software Development Kit
- Android Development Tools plug-in (from within Eclipse)

In the subsections that follow, we will cover these stages in detail for a PC running Windows (tested on XP, Vista, and Windows 7).

Installing the JDK (Java Development Kit)

Go to the Java download page at *http://www.oracle.com/technetwork/java/javase/downloads/index.html*.

Select the Java icon to access the JDK downloads:

The list of JDK downloads will be shown. Click the Accept License Agreement radio button; otherwise, you will not be allowed to continue. Download and run the latest JDKs present; as of this writing, they are *jdk-7u2-windows-i586.exe* (or *jdk-7u2-windows-x64.exe* for 64-bit Windows). You may need to select the location of the download site. Accept any security warnings that appear, but only if you are downloading from the official Java download web page.

When the download has completed and is run you will need to go through the install screens, clicking Next until the JDK installer has finished. You should not need to change any options presented. When the JDK installer has completed, click the Finish button. A product registration web page may load; you can close this or you can choose to register your installation.

Installing Eclipse for Java development

The Eclipse Downloads web page is at *http://www.eclipse.org/downloads/*.

Windows needs to be selected in the Packages drop down; select the relevant Eclipse IDE for Java Developers download link (see Figure 1-8).

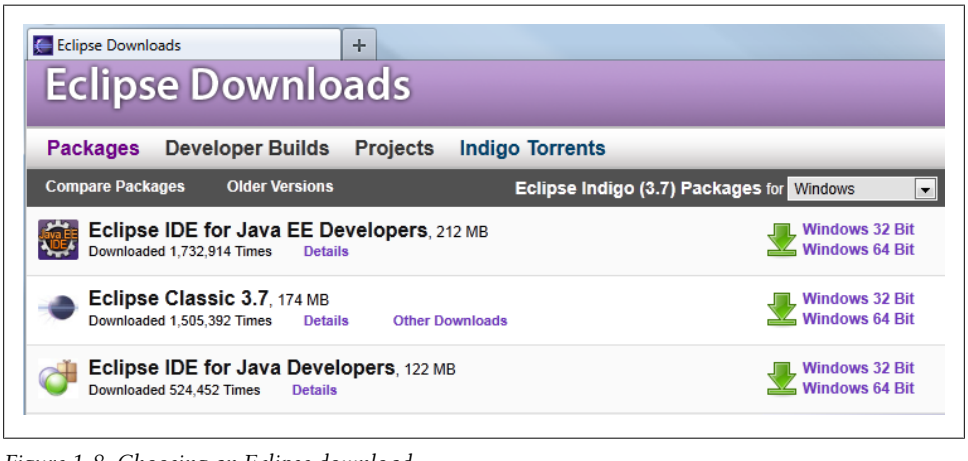


Figure 1-8. Choosing an Eclipse download

Download and open the ZIP file. In the file there will be an *eclipse* directory containing several files and subdirectories. Copy the *eclipse* directory and all its contents as it comes (Figure 1-9). The usual place to copy the files to is either the root of the C drive or under *C:\Program Files*. You may need to select Continue when Windows asks permission for the copy.

Make a desktop shortcut to *eclipse.exe*.

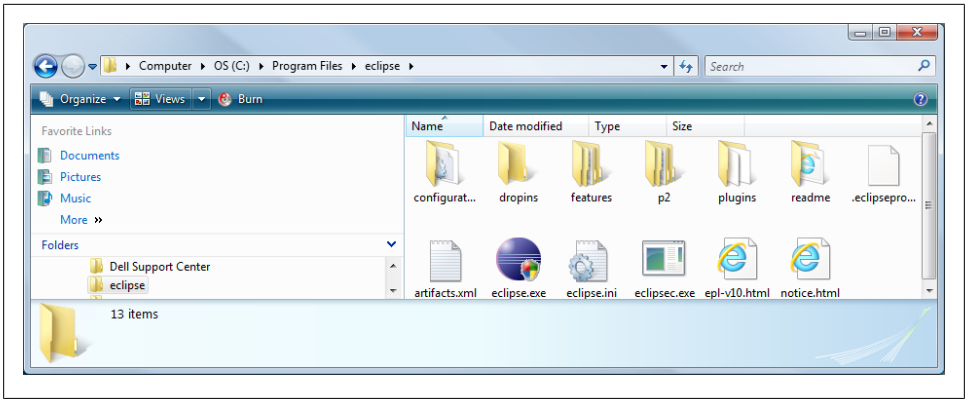
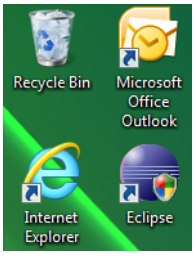


Figure 1-9. Contents of the Eclipse folder



Run Eclipse so that it sets up a workspace; this will also check that both Java and Eclipse were installed correctly. When you run Eclipse a security warning may be displayed; select Run to continue. Accept the default workspace location or use a different directory.

Installing the Android SDK (software development kit)

Go to the Android Software Development Kit download page at *http://developer.android.com/sdk/index.html*.

Choose the latest Windows EXE package (currently *installer_r16-windows.exe*) and select Run. Accept the security warning only if you are downloading from the official Android SDK website. The Android SDK Tools installer will show some screens. Select the Next button on each screen; you should not need to change any options. Since *C:\Program Files* is a protected directory, you can either get permission to install there or, as some developers do, install to your user folder or another directory—for example, *C:\Android\android-sdk*.

When the Install button is clicked, a progress screen will briefly display while the Android files are copied. Click the final Next button and the Finish button at the end of the installation. If you left the Start SDK Manager checkbox ticked the SDK Manager will run. Otherwise, select SDK Manager from the Android SDK Tools program group (Start→All Programs→Android SDK Tools→SDK Manager). When the SDK Manager

starts the Android packages available to download are checked. Then a list of all available packages is shown with some preselected for download. A Status column shows whether a package is installed or not. In Figure 1-10, you can see that the Android SDK Tools have just been installed and this is reflected in the Status column.

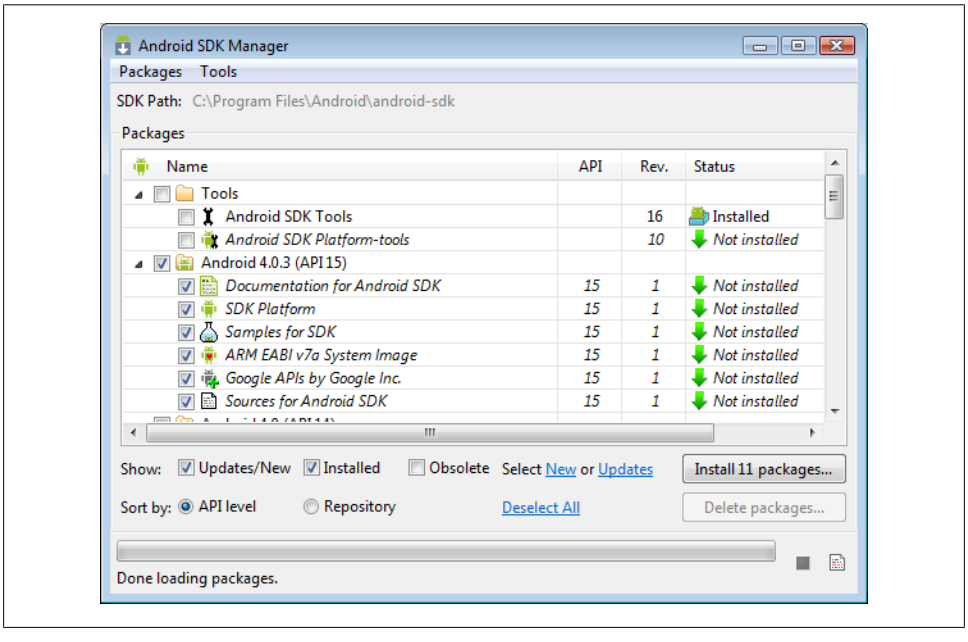

Android SDK Manager
Packages Tools
SDK Path: C:\Program Files\Android\android-sdk
Packages
Name API Rev. Status
Tools
Android SDK Tools 16 Installed
Android SDK Platform-tools 10 Not installed
Android 4.0.3 (API 15)
Documentation for Android SDK 15 1 Not installed
SDK Platform 15 1 Not installed
Samples for SDK 15 1 Not installed
ARM EABI v7a System Image 15 1 Not installed
Google APIs by Google Inc. 15 1 Not installed
Sources for Android SDK 15 1 Not installed
Show: Updates/New Installed Obsolete Select New or Updates
Install 11 packages...
Sort by: API level Repository
Deselect All
Delete packages...
Done loading packages.

Figure 1-10. Android SDK Manager, showing installed and downloadable components

Check each package that needs to be installed. Multiple packages are available. These include SDK platform packages for each application programming interface (API) level, application samples for most API levels, Google Maps APIs, manufacturer-device-specific APIs, documentation, source code, and the following Google extra packages:

Android Support
: Used to support later Android APIs on older devices

AdMob Ads SDK
: For incorporating advertising into apps

Analytics SDK
: To support analysis of customers' purchases

Market Billing
: Adds support for in-app purchases

Market Licensing
: Helps protect apps from being illegally copied

USB Driver
: For debugging on physical devices (or using a manufacturer's driver)

Webdriver
: Helps test a website's compatibility with the Android browser

It is recommended that you download several SDK platforms to allow testing of apps against various device configurations. It is worth noting that older computers will struggle to run the virtual device emulators for the later Android APIs; therefore, develop with the earlier SDK platforms on such computers. If in doubt about what to download, either accept the initial choices and rerun the SDK Manager to get other packages as and when required; or check all packages to download everything (the download may take a while). Click the "Install packages" button.

The selected packages will be shown in a list; if a package has licensing terms that require acceptance, it is shown with a question mark. Highlight each package that has a question mark to read the licensing terms. You can accept or reject the package using the radio buttons. Rejected packages are marked with a red ×. Alternatively, click Accept All to accept everything that is available. Click the Install button and a progress log will show the packages being installed, as well as any errors that occur. On Windows a common error occurs when the SDK Manager is unable to access or rename directories. Rerun the SDK Manager as administrator and check that the directory does not have any read-only flags or files; see Recipe 1.12 for further details. When complete close the SDK Manager by clicking the × button in the top corner of the window.

Installing the Android Development Tools (ADT) plug-in

You install the ADT plug-in via Eclipse, but to do so you must run Eclipse from the administrator account. Use the shortcut created earlier or *eclipse.exe* from the *eclipse* folder. In either case, bring up the context menu (usually via a right-click), select "Run as administrator," and accept any security warnings. When Eclipse has loaded open the Help menu item and select Install New Software....

On the Install screen enter the following address in the "Work with" box:

https://dl-ssl.google.com/android/eclipse/

Click the Add button. An Add Repository screen appears; in the Name box type something meaningful, such as "ADT plug-in" (the aforementioned web address will be displayed in the Location box); see Figure 1-11.

Click the OK button. The screen will update after briefly showing Pending in the Name column of the table.

Check the box next to Developer Tools. Then select the Next button at the bottom of the screen (see Figure 1-12).

A list of the items to be installed will be displayed. If you get an error message check that Eclipse has been run under the administrator account. Select Next again. A screen displays the licenses; ensure that each license has been accepted (select the "I accept the terms of the license agreements" radio button). Then click the Finish button. A

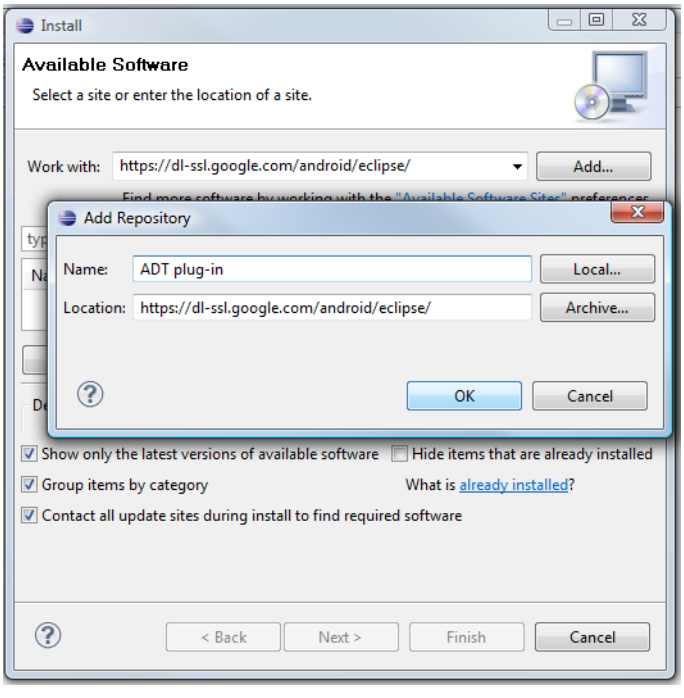


Figure 1-11. Adding the ADT plug-in repository

security warning will need to be accepted to complete the installation; select OK to this warning (the address entered earlier is a secure address). Eclipse will ask you for a restart. Select the Restart Now button and Eclipse will close and reload. A Welcome to Android Development dialog will appear. Set the SDK location in the Existing Location box (since the SDK Manager will have already run), browse to the Android SDK folder (by default, *C:\Program Files\Android\android-sdk*), and click Next (see Figure 1-13).

A Google Android SDK usage monitoring question will appear; change the option if required and click Finish. Eclipse is now configured to build and debug Android apps. See Recipe 3.3 to configure an Android emulator; then try Recipe 1.4 as a sanity check. Plug a physical device into the computer and use its settings to turn on USB Debugging (under Development in Applications).

See Also

Recipe 1.4; Recipe 1.12; Recipe 3.3; *http://developer.android.com/sdk/installing.html*, *http://www.eclipse.org/*; *http://www.oracle.com/technetwork/java/javase/downloads/index.html*

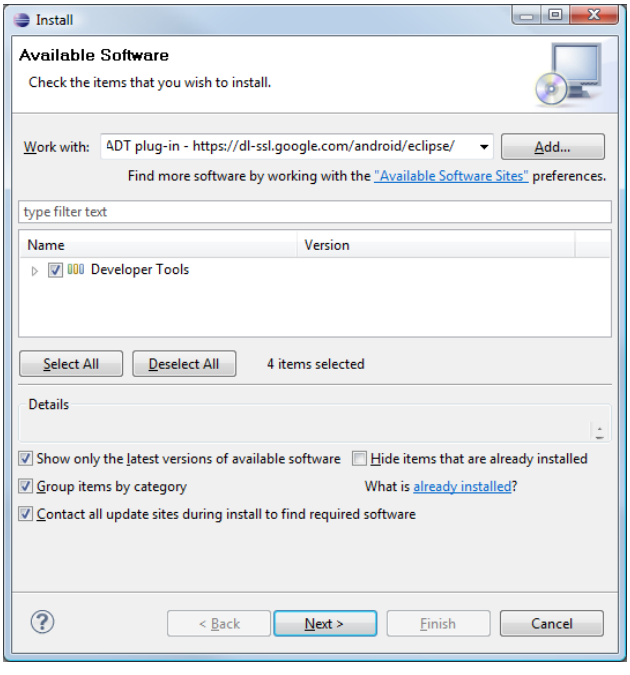


Figure 1-12. Choosing what to install

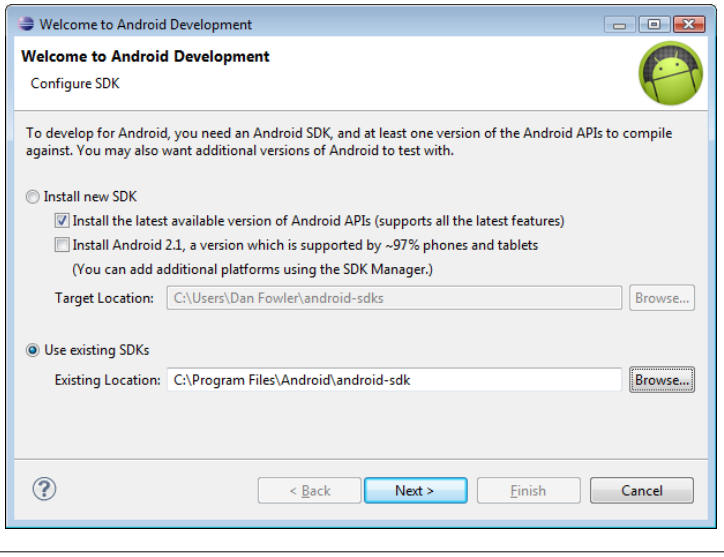

Welcome to Android Development
Welcome to Android Development
Configure SDK
To develop for Android, you need an Android SDK, and at least one version of the Android APIs to compile against. You may also want additional versions of Android to test with.
Install new SDK
Install the latest available version of Android APIs (supports all the latest features)
Install Android 2.1, a version which is supported by ~97% phones and tablets
(You can add additional platforms using the SDK Manager.)
Target Location:
C:\Users\Dan Fowler\android-sdks
Browse...
Use existing SDKs
Existing Location:
C:\Program Files\Android\android-sdk
Browse...
< Back
Next >
Finish
Cancel

Figure 1-13. Connecting the newly installed SDK to the newly installed ADT plug-in

1.6 Understanding the Android Life Cycle

Ian Darwin

Problem

Android apps do not have a "main" method; you need to learn how they get started and how they stop or get stopped.

Solution

The class `android.Activity` provides a number of well-defined life-cycle methods that are called when an application is started, suspended, restarted, and so on, as well as a method you can call to mark an activity as finished.

Discussion

Your Android application runs in its own Unix process, so in general it cannot directly affect any other running application. The Dalvik VM interfaces with the operating system to call you when your application starts, when the user switches to another application, and so on. There is a well-defined life cycle for Android applications.

An Android application has three states it can be in:

- Active, in which the app is visible to the user and is running
- Paused, in which the app is partly obscured and has lost the input focus
- Stopped, in which the app is completely hidden from view

Your app will be transitioned among these states by Android calling the following methods on the current activity at the appropriate time:

```
void onCreate(Bundle savedInstanceState)
void onStart()
void onResume()
void onRestart()
void onPause()
void onStop()
void onDestroy()
```

You can see the state diagram for this life cycle in Figure 1-14.

For an application's first activity, `onCreate()` is how you know that the application has been started. This is where you normally do constructor-like work such as setting up the "main window" with `setContentView()`, adding listeners to buttons to do work (including starting additional activities), and so on. This is the one method that even the simplest Android app needs.

You can see the effects of the various life cycle methods by creating a dummy project in Eclipse and overriding all the methods with log "debug" statements.

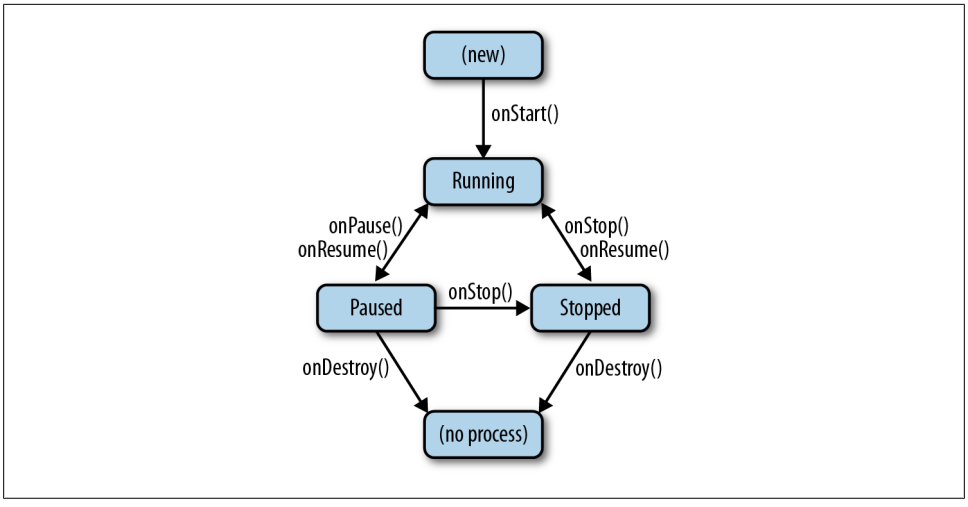


Figure 1-14. Android life-cycle states

1.7 Installing .apk Files onto an Emulator via the ADB

Rachee Singh

Problem

You have an application's *.apk* file, and you want to install it on the emulator to check out the application, or because an application you are developing requires it.

Solution

Use the ADB command-line tool to install the *.apk* file onto the running emulator; you can also use this tool to install an *.apk* file onto a connected Android device.

Discussion

To install the *.apk* file, follow these steps:

1. Find the location on your machine where you have installed the Android SDK. In the Android SDK directory, go to the *tools* directory.
2. Look for an executable named adb in the *tools* directory. If it is present that is the location of the adb file; otherwise, there should be a *.txt* file named "adb has moved." The contents of the file merely direct you to the location of the adb binary; the file states that adb is present in the *platform-tools* directory instead of the *tools* directory.
3. Once you have located the adb program, `cd` to that location in a terminal (Linux) or command prompt (Windows).

4. Use the command `adb install` *`location of the .apk you want to install`*. If you get "command not found" on Linux, try using "./adb" instead of just "adb".

This should start the installation on the device that is currently running (either an emulator that is running on your desktop, or a physical Android device that is connected).

After the installation finishes, in the menu of the Android device/emulator you should see the icon of the application you just installed (see Figure 1-15).

Figure 1-15. The installation command

1.8 Installing Apps onto an Emulator via SlideME

David Dawes

Problem

App stores are a huge element of the attraction of modern smartphones. Google's Android Market is the official app store, but you may want to use others as well.

Solution

SlideMe LLC offers an alternative app store. The SlideME app store allows you to install other apps (perhaps you want to integrate with other apps), as well as test the experience of publishing and downloading your own apps on your emulated Android device. SlideME also reaches many Android users who are locked out of the Google Android Market, including people with unsupported devices and those who don't live in a country that is supported by the Android Market.

Discussion

An alternative to the official Android Market is Slide ME, an alternative app store. SlideME may not have as many apps as Google's Android Market, but it has some advantages, including that it works easily on an emulated Android device.

Go to the SlideME website using your emulated Android device, browse or search through the apps, and click on a free one. After a pause to download the file, open the download (the little arrow on the top left), review the license, and launch the *.apk* file you've downloaded to install the app. During the installation, you will be asked to review and accept the license for the software.

Once the SlideME app is installed, you can go through the catalog and install more apps without using the browser. This is much easier than using a web browser to download the apps, since the presentation is designed for the Android device; simply choose a category, scroll through it, and choose an app to install. I have had some stability problems using the app on my emulator—it freezes on occasion—but I was able to install some basic free apps, like Grocery List.

I noticed in the Android Invasion discussion forum on Linkedin.com that some Android users are disappointed to find that many cell phone providers do *not* include the official Android Market in their Android cell phone offerings, and unless you're comfortable rooting and flashing your Android phone there's no way to get it. Most consumers are not comfortable rooting and flashing their phones, and for them SlideME offers an alternative way to find free and inexpensive apps for their phones.

See Also

SlideME also allows you to publish your apps to its app store; see the Applications page on the SlideME website.

For information on developing apps for SlideME, see *http://slideme.org/developers*.

1.9 Sharing Java Classes from Another Eclipse Project

Ian Darwin

Problem

You want to use a class from another project, but you don't want to copy and paste.

Solution

Add the project as a "referenced project," and Eclipse (and DEX) will do the work.

Discussion

You often need to reuse classes from another project. In my JPSTrack GPS tracking program, the Android version borrows classes such as the file I/O module from the Java SE version. You surely do not want to copy and paste classes willy-nilly from one project into another, because this makes maintenance improbable.

In the simplest case, when the library project contains the source of the classes you want to import, all you have to do is declare the project containing the needed classes (the Java SE version in this case) as a referenced project on the build path. Select Project→Properties→Java Build Path, select Projects, and click Add. In Figure 1-16, I am adding the SE project "jpstrack" as a dependency on the Android project "jpstrack.android."

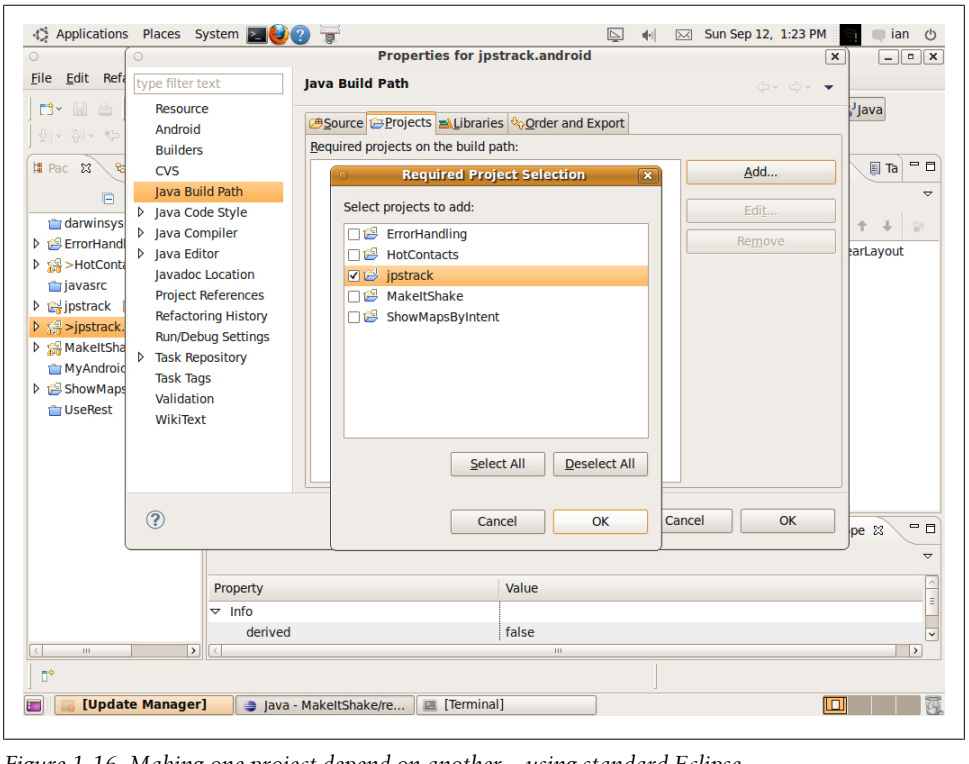


Figure 1-16. Making one project depend on another—using standard Eclipse

Mobile developers who create apps for other platforms as well should note that this technique does not work if you also have the current (late 2011) BlackBerry Java plug-in installed in your Eclipse installation. This is a bug in the BlackBerry Java plug-in; it incorrectly flags even non-BlackBerry projects as depending on non-BlackBerry-library projects, and marks the project as having an error, which will prevent correct code

generation and execution. Remove the buggy plug-in, or put it in its own Eclipse installation.

Alternatively, create a JAR file using either Ant or the Eclipse wizard. Have the other project refer to it as an external JAR in the classpath settings. Or physically copy it into the *libs* directory and refer to it from there.

A newer method that is often more reliable and is now officially recommended, but is only useful if both projects are Android projects, is to declare the library one as a library project, under Project→Properties→Android→Library tab, and use the Add button on the other project on the same screen to list the library project as a dependency on the main project (see Figure 1-17).

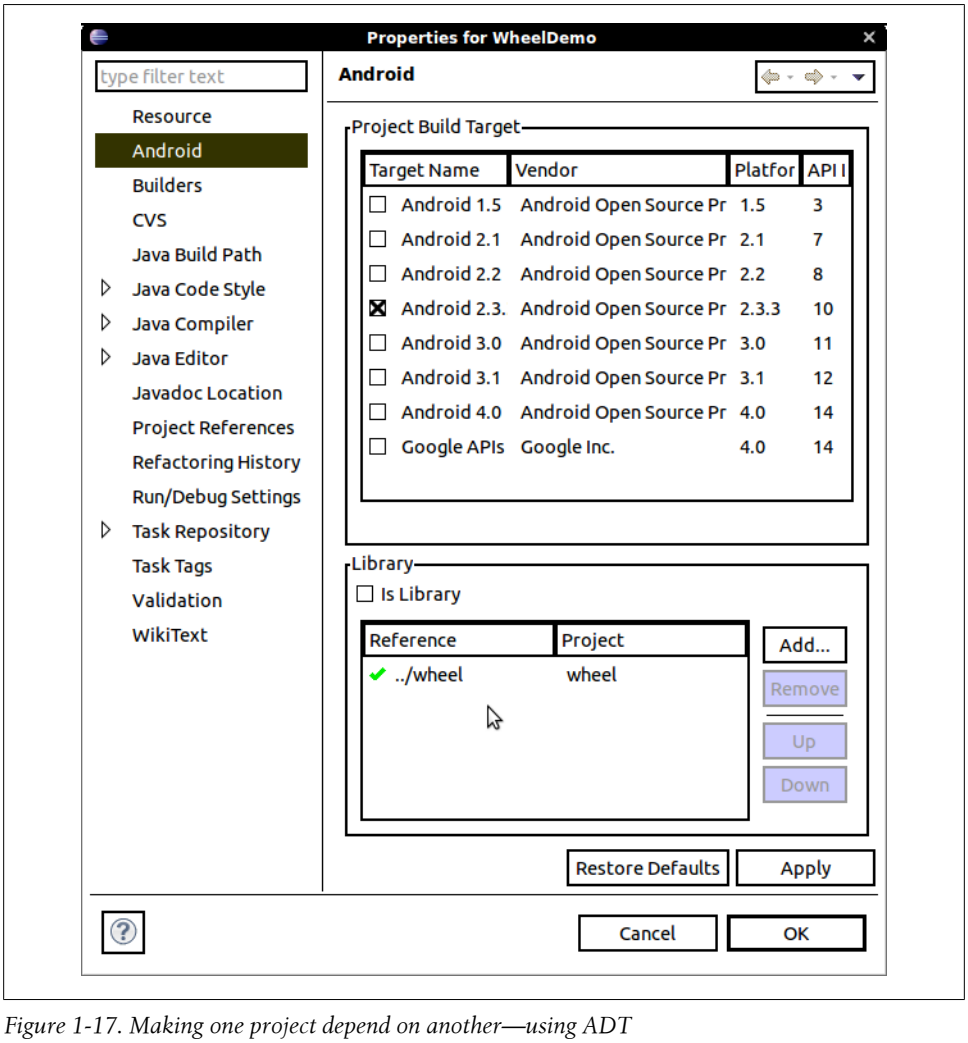


Figure 1-17. Making one project depend on another—using ADT

For command-line fans, the first method involves editing the *.classpath* file, while the second method simply creates entries in the *project.properties* file, for example:

```
# Project target
target=android-7
android.library=false
android.library.reference.1=../wheel
```

Since you are probably keeping both projects under source control (and if these are programs you ever intend to ship, you should!), remember to "tag" both projects when you release the Android project—one of the points in favor of source control is that you are able to re-create exactly what you shipped.

See Also

See the official documentation on Library Projects.

1.10 Referencing Libraries to Implement External Functionality

Rachee Singh

Problem

You need to reference an external library in your source code.

Solution

Obtain the JAR file for the library that you require and add it to your project.

Discussion

As an example, you might need to use AndroidPlot, a library for plotting charts and graphs in your application, or OpenStreetMap, a wiki project that creates and provides free geographic data and mapping. If so, your application needs to reference these libraries. You can do this in Eclipse in a few simple steps:

1. Download the JAR file corresponding to the library you wish to use.
2. After creating your Android project in Eclipse, right-click on the project name and select Properties in the menu (Figure 1-18).
3. From the list on the left side, select Java Build Path and click on the Libraries tab.
4. Click the Add External JARs button.
5. Provide the location where you downloaded the JAR file for the library you wish to use.

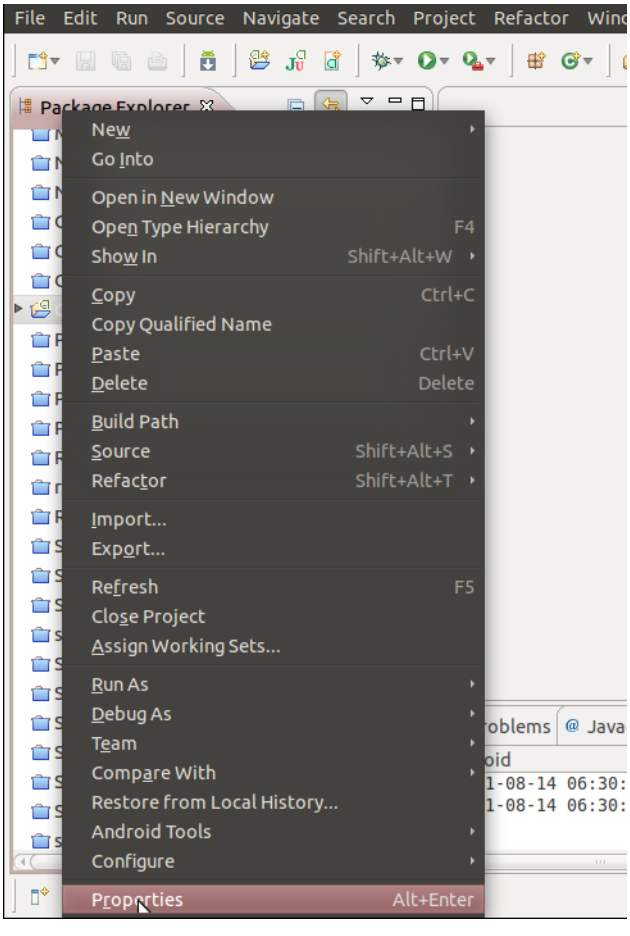


Figure 1-18. Selecting project properties

At this point you will see a *Referenced Libraries* directory in your project. The JARs you added will appear (see Figure 1-19).

An alternative approach is to create a *lib* folder in your project, physically copy the JAR files there, and add them individually as you did earlier, but instead clicking the Add JARs button. This keeps everything in one place (especially if your project is shared via a version control system with others who might use a different operating system and be unable to locate the external JARs in the same place). However, it does raise the burden of responsibility for licensing issues on the included JAR files. See Figure 1-20.

In either case, if you also build with Ant, be sure to update your *build.xml* file.

Whichever way you do it, it's pretty easy to add libraries to your project.

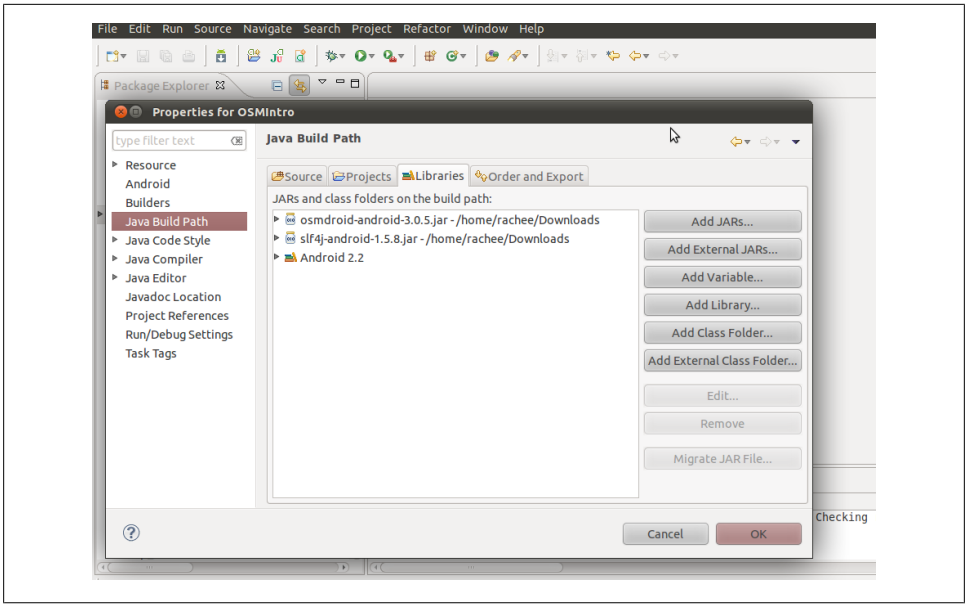


Figure 1-19. Adding libraries

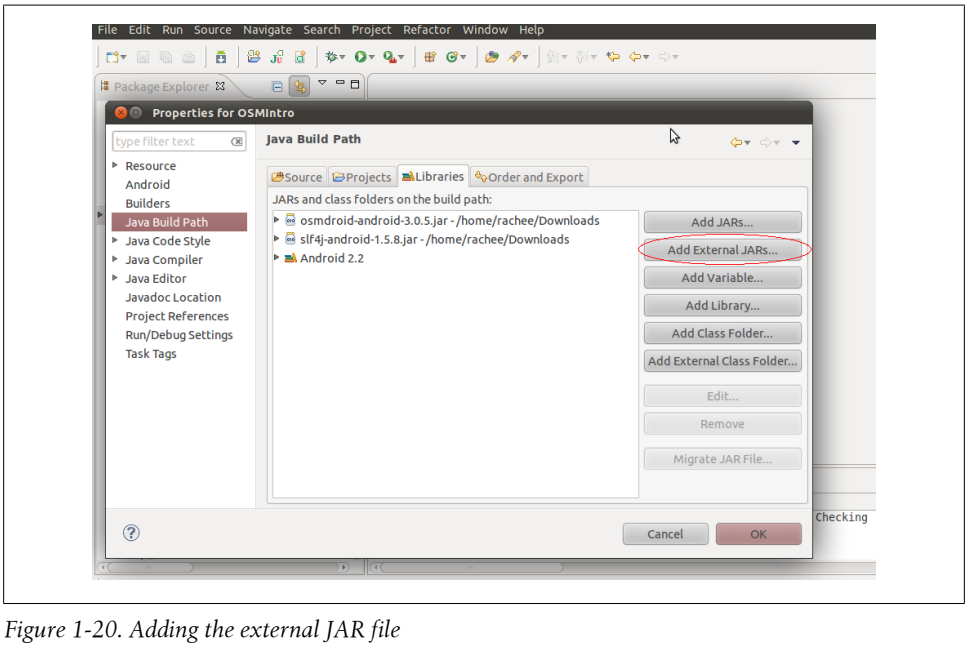


Figure 1-20. Adding the external JAR file

1.11 Using SDK Samples to Help Avoid Head Scratching

Daniel Fowler

Problem

Sometimes it is a struggle to code up some functionality, especially when the documentation is sketchy or does not provide any examples.

Solution

Looking at existing working code will help. The Android SDK has sample programs that you can pick apart to see how they work.

Discussion

The Android SDK comes with several sample applications that can be useful when trying to code up some functionality. Looking through the sample code can be insightful. Once you have installed the Android SDK, several samples become available:

- Accelerometer Play
- Accessibility Service
- API Demos
- Backup and Restore
- Bluetooth Chat
- Business Card
- Contact Manager
- Cube Live Wallpaper
- Home
- Honeycomb Gallery
- JetBoy
- Lunar Lander
- Multiple Resolutions
- Near Field Communication
- Note Pad
- RenderScript
- Sample Sync Adapter
- Searchable Dictionary
- Session Initiation Protocol
- Snake
- Soft Keyboard

- Spinner
- SpinnerTest
- StackView Widget
- TicTacToeLib
- TicTacToeMain
- USB
- Wiktionary
- Wiktionary (Simplified)
- Weather List Widget
- XML Adapters

To open a sample project from Eclipse open the File menu and then select Android Project. See Figure 1-21.

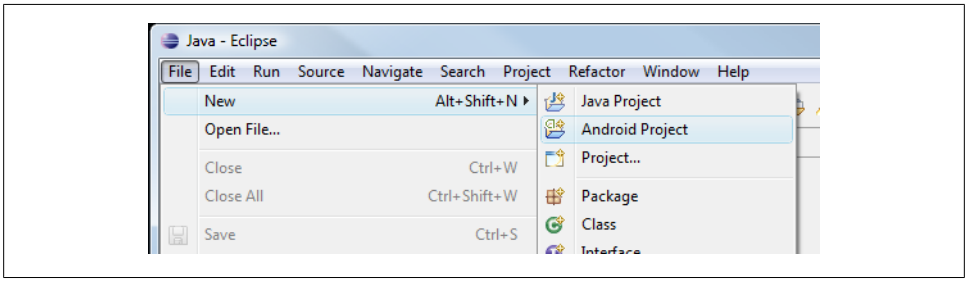
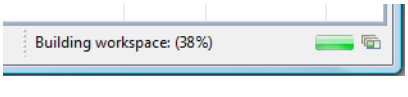


Figure 1-21. Starting a new Android project

On the New Android Project dialog, select the "Create project from existing sample" option. Click Next and select the Build Target. A list of available samples for the selected target is shown. If the required sample is not shown, go back and select another Build Target. (The sample may not be installed; the SDK Manager can be used to install additional samples if they were missed during the SDK setup.) Choose the sample to load, click Finish, and the sample is copied to the Workspace and built (with progress shown on the status bar).



After a short time, the sample will be ready to run and you will be able to browse the source code to see how it is all done.

If the samples have been moved from the SDK *samples* directory, use the "Create project from existing source" option on the New Android Project dialog to open the sample.

When the sample is first run select Android Application in the Run As dialog that may appear. It may also be necessary to configure an appropriate AVD to run the sample (see Recipe 3.3). See Figure 1-22.

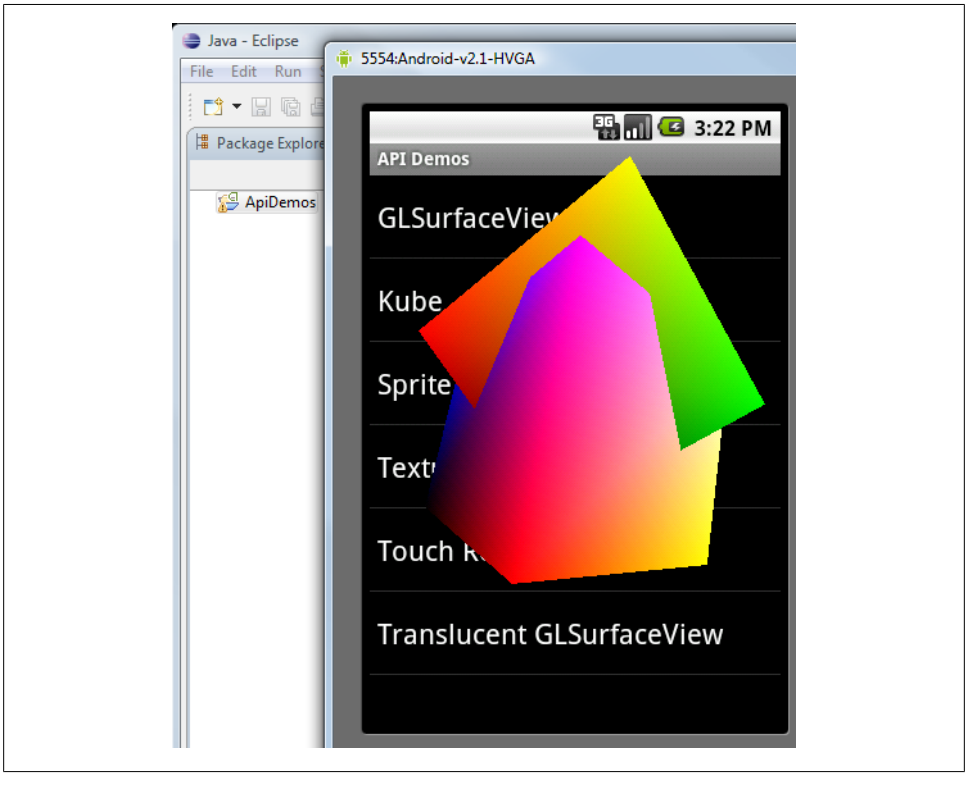


Figure 1-22. API demos in action

See Also

The Android Developers website at *http://developer.android.com/index.html*; this cookbook, of course.

You can also search the Web for additional programs or examples. If you still can't find what you need, you can seek help from Stack Overflow (*http://www.stackoverflow.com*; use "android" as the tag) or from the Internet Relay Chat (IRC) channel #android-dev on freenode.

1.12 Keeping the Android SDK Updated

Daniel Fowler

Problem

The SDK must be kept updated to allow app developers to work with the latest APIs on the evolving Android platform.

Solution

Use the Android SDK Manager program to update the existing installed SDK packages and to install new SDK packages. This includes third-party packages for device-specific functionality.

Discussion

The Android operating system (OS) is constantly evolving, and therefore, so is the Android SDK. The ongoing development of Android is driven by:

- Google's research and development
- Phone manufacturers developing new and improved handsets
- Addressing security issues and possible exploits
- The need to support new devices (e.g., support for tablet devices was added with version 3.0)
- Support for new hardware interfaces (e.g., support for near field communication was added in version 2.3).
- Fixing bugs
- Improvements in functionality (e.g., a new JavaScript engine)
- Changes in the underlying Linux kernel
- Deprecation of redundant programming interfaces
- New uses (e.g., Google TV)
- The wider Android development community

We covered Android SDK installation elsewhere (see Recipe 1.5 or *http://developer.android.com/sdk/installing.html*). After the SDK is installed on the development machine and the programming environment is running smoothly, once in a while developers will need to check for updates to the SDK.

You can keep the SDK up-to-date by running the SDK Manager program. (On a Windows machine run *SDK Manager.exe* in the folder *C:\Program Files\Android\android-sdk*, or use the Start button, then select All Programs→Android SDK Tools, and click SDK Manager). You can also run it from within Eclipse (using the Window menu and selecting Android SDK Manager). See Figure 1-23. The Android SDK is divided into

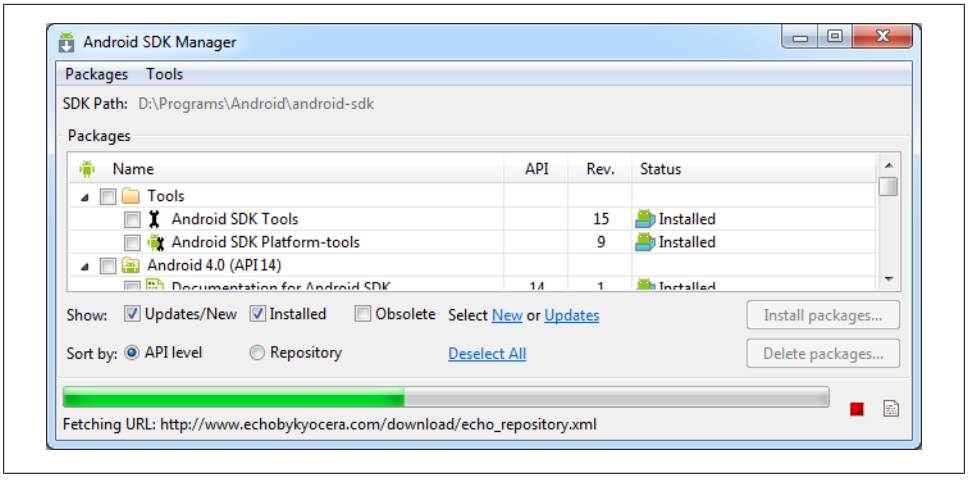


Figure 1-23. The Android SDK Manager

several packages. The SDK Manager automatically scans for updates to existing packages and will list new packages and those provided by device manufacturers.

Available updates will be shown in a list (as will available optional packages). If an update or package has licensing terms that require acceptance it is shown with a question mark. Highlight each package that has a question mark to read the licensing terms. You can accept or reject the package using the radio buttons. Rejected packages are marked with a red ×. See Figure 1-24.

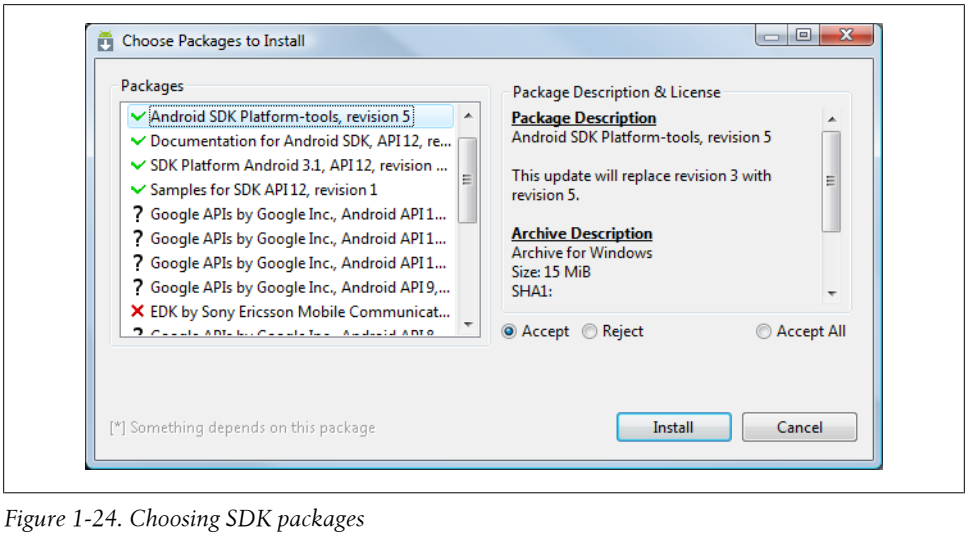

Choose Packages to Install
Packages
Android SDK Platform-tools, revision 5
Documentation for Android SDK, API 12, re...
SDK Platform Android 3.1, API 12, revision ...
Samples for SDK API 12, revision 1
Google APIs by Google Inc., Android API 1...
Google APIs by Google Inc., Android API 1...
Google APIs by Google Inc., Android API 1...
Google APIs by Google Inc., Android API 9,...
EDK by Sony Ericsson Mobile Communicat...
Package Description & License
Package Description
Android SDK Platform-tools, revision 5
This update will replace revision 3 with revision 5.
Archive Description
Archive for Windows
Size: 15 MiB
SHA1:
Accept Reject Accept All
[*] Something depends on this package
Install Cancel

Figure 1-24. Choosing SDK packages

Alternatively, click on Accept All to accept everything that is available. All packages and updates that are ready to download and install will be shown with a green tick.

Click the Install button to begin the download and installation; when complete click the Close button. See Figure 1-25.

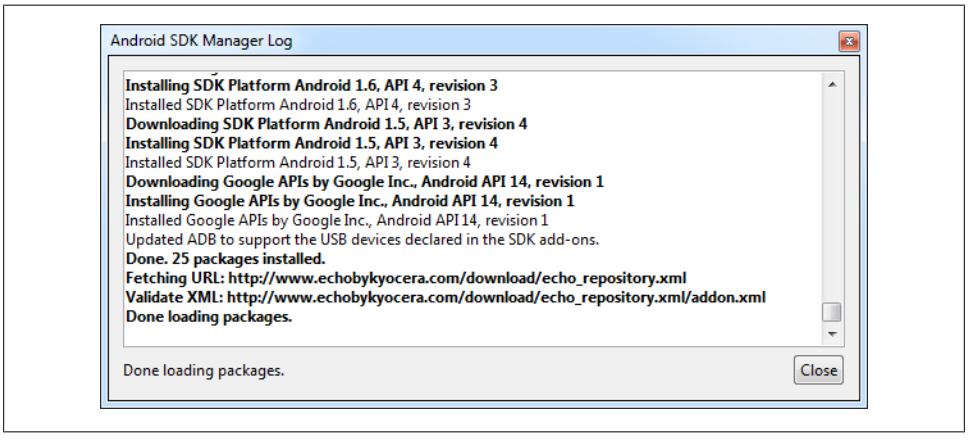

Android SDK Manager Log
Installing SDK Platform Android 1.6, API 4, revision 3
Installed SDK Platform Android 1.6, API 4, revision 3
Downloading SDK Platform Android 1.5, API 3, revision 4
Installing SDK Platform Android 1.5, API 3, revision 4
Installed SDK Platform Android 1.5, API 3, revision 4
Downloading Google APIs by Google Inc., Android API 14, revision 1
Installing Google APIs by Google Inc., Android API 14, revision 1
Installed Google APIs by Google Inc., Android API 14, revision 1
Updated ADB to support the USB devices declared in the SDK add-ons.
Done. 25 packages installed.
Fetching URL: http://www.echobykyocera.com/download/echo_repository.xml
Validate XML: http://www.echobykyocera.com/download/echo_repository.xml/addon.xml
Done loading packages.
Done loading packages.
Close

Figure 1-25. SDK Manager Log window

If the SDK Manager program has itself been updated, you will see a message asking you to restart the program (see Figure 1-26).

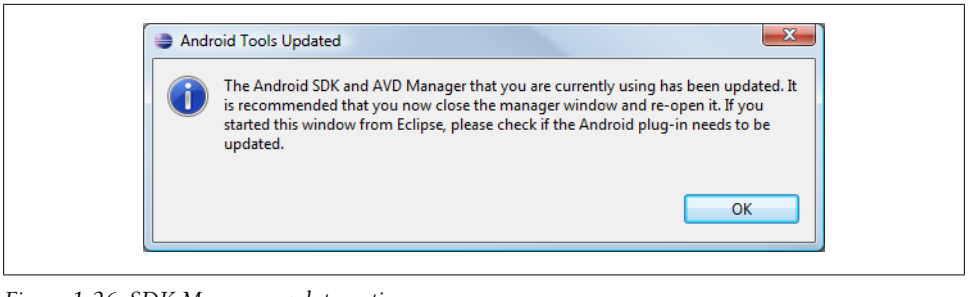

Android Tools Updated
The Android SDK and AVD Manager that you are currently using has been updated. It is recommended that you now close the manager window and re-open it. If you started this window from Eclipse, please check if the Android plug-in needs to be updated.
OK

Figure 1-26. SDK Manager update notice

The SDK Manager is also used to download additional packages that are not part of the standard platform. This mechanism is used by device manufacturers to provide support for their own hardware. For example, LG Electronics provides a 3D device, and to support 3D capability in applications an additional package is provided. It is also used by Google to allow the download of optional APIs.

In the SDK Manager dialog, expand and tick the required packages in the left-hand list, and then click the Install button (see Figure 1-27). If a third-party package is not listed, the URL to a *respository.xml* file, provided by the package publisher, will need to be entered via the Tools menu.

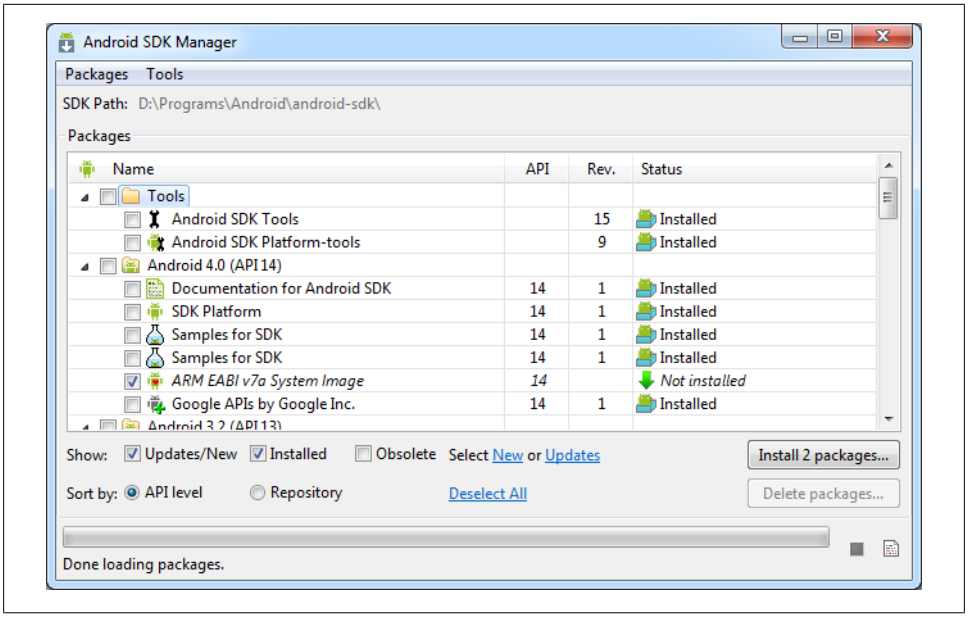

Android SDK Manager
Packages Tools
SDK Path: D:\Programs\Android\android-sdk\
Packages
Name API Rev. Status
Tools
Android SDK Tools 15 Installed
Android SDK Platform-tools 9 Installed
Android 4.0 (API 14)
Documentation for Android SDK 14 1 Installed
SDK Platform 14 1 Installed
Samples for SDK 14 1 Installed
Samples for SDK 14 1 Installed
ARM EABI v7a System Image 14 Not installed
Google APIs by Google Inc. 14 1 Installed
Show: Updates/New Installed Obsolete Select New or Updates
Install 2 packages...
Sort by: API level Repository
Deselect All
Delete packages...
Done loading packages.

Figure 1-27. List of installed and installable components

Possible update errors on Windows

In a system this complex, there are many things that might go wrong. This section discusses some of these and their solutions.

Run SDK Manager as admin. On a Windows machine, the default location for the SDK is under the *C:\Program Files\Android\android-sdk* directory. This is a restricted directory and can cause the SDK installation to fail. A message dialog with the title "SDK Manager: failed to install" can appear (see Figure 1-28).

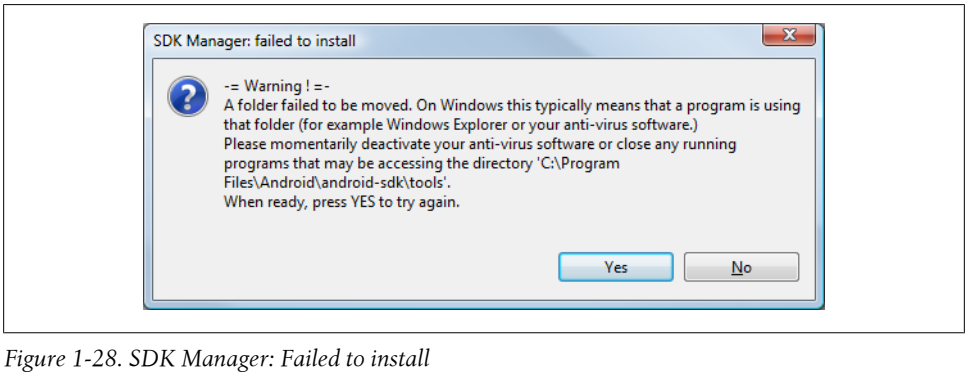

SDK Manager: failed to install
-= Warning ! =-
A folder failed to be moved. On Windows this typically means that a program is using that folder (for example Windows Explorer or your anti-virus software.)
Please momentarily deactivate your anti-virus software or close any running programs that may be accessing the directory 'C:\Program Files\Android\android-sdk\tools'.
When ready, press YES to try again.
Yes
No

Figure 1-28. SDK Manager: Failed to install

To overcome this error there are a few things to check:

- Unplug any Android devices (this may prevent *adb.exe* from closing).
- Browse to *C:\Program Files\Android\Android-sdk* and bring up the Properties for the *tools* folder (select the context menu, and then Properties). Ensure that the "Read-only (Only applies to files in folder)" checkbox is cleared (see Figure 1-29).

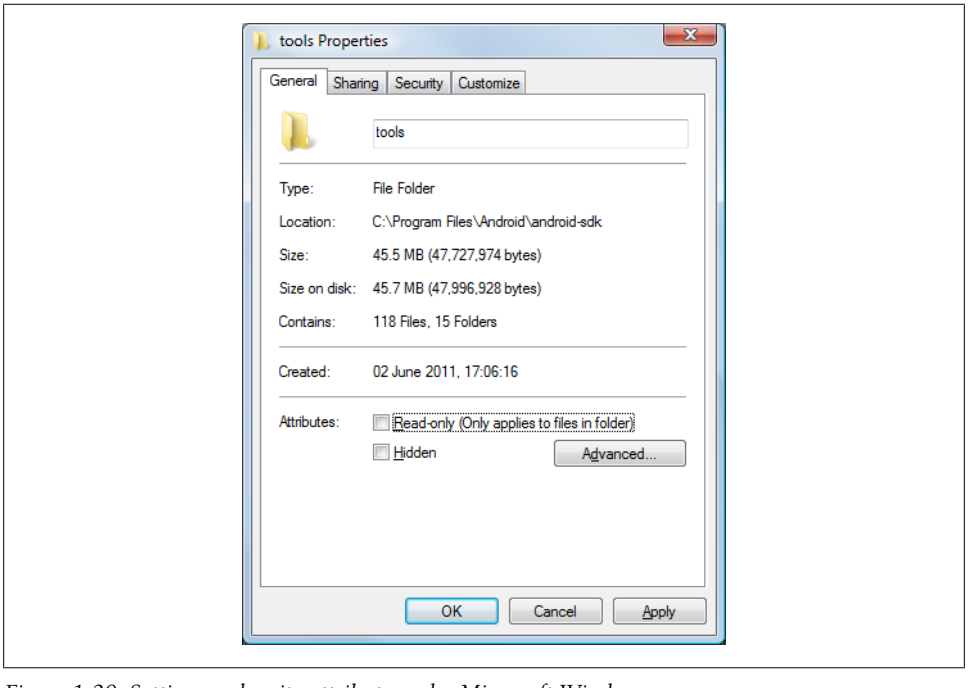


Figure 1-29. Setting read-write attribute under Microsoft Windows

You may need to give permission to change the attributes (see Figure 1-30).

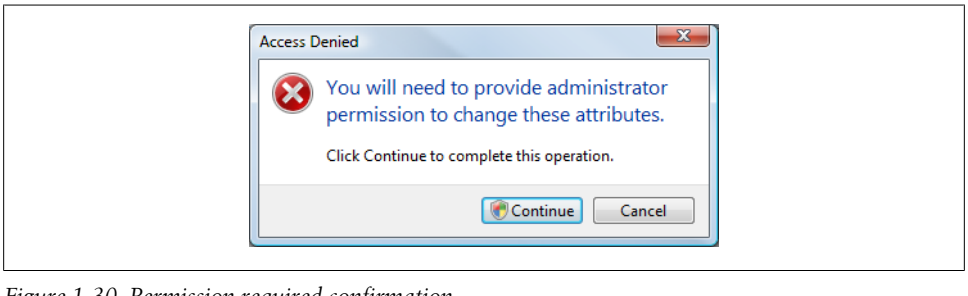

Access Denied
You will need to provide administrator permission to change these attributes.
Click Continue to complete this operation.
Continue
Cancel

Figure 1-30. Permission required confirmation

A Confirm Attribute Changes dialog will appear; ensure the option "Apply changes to this folder, subfolders and files" is selected and click OK. Then do the following:

- Restart the computer.
- Ensure that all other programs are closed, especially any copies of File Explorer.
- Run *SDK Manager.exe* under the administrator account. Bring up the context menu and select "Run as administrator. (See Figure 1-31.)

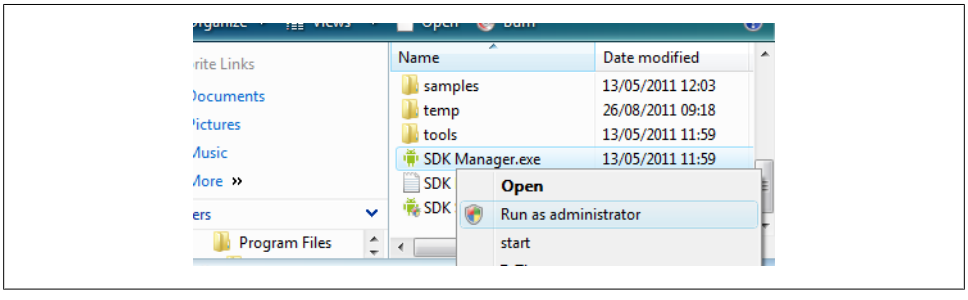


Figure 1-31. Run as administrator

Close ADB before updating. A message asking you to restart ADB (the Android Debugger) may appear (Figure 1-32).

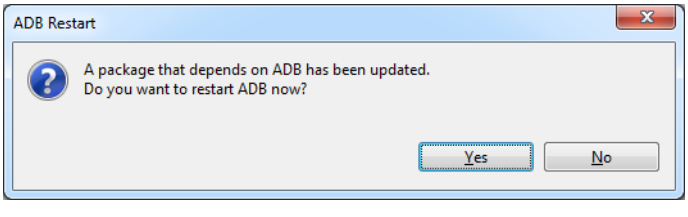


Figure 1-32. Confirmation to restart ADB

Ideally, it is best to run the SDK Manager without ADB running, and it should not be running if Windows has just been started. Alternatively, you can use the Windows Task Manager to stop *adb.exe*. Answer No to this prompt if ADB was not running; otherwise, answer Yes.

SDK Manager cannot update itself. During the SDK update installation there may be an error related to the SDK Manager program (see Figure 1-33).

To resolve this error ensure that all programs are closed (including *adb.exe*). Then copy *SDK Manager.exe* from *C:\Program Files\Android\android-sdk\tools\lib* to *C:\Program Files\Android\android-sdk* (or wherever the SDK is installed). Then run the SDK Manager again. (See Figure 1-32.)

Updating Eclipse. After you update the SDK and open Eclipse a warning message may appear (see Figure 1-34).

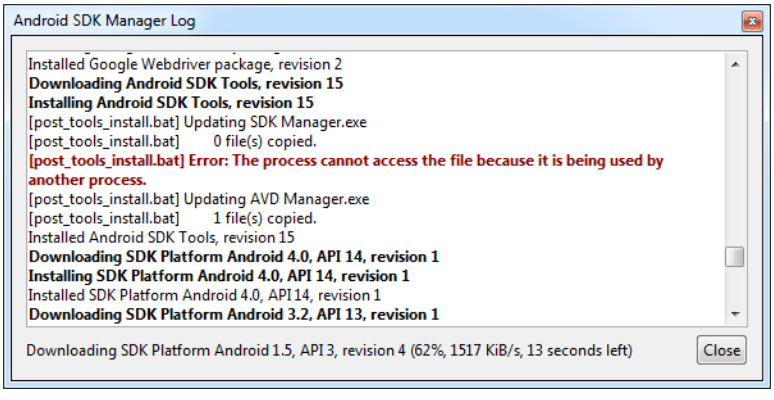

Android SDK Manager Log
Installed Google Webdriver package, revision 2
Downloading Android SDK Tools, revision 15
Installing Android SDK Tools, revision 15
[post_tools_install.bat] Updating SDK Manager.exe
[post_tools_install.bat] 0 file(s) copied.
[post_tools_install.bat] Error: The process cannot access the file because it is being used by another process.
[post_tools_install.bat] Updating AVD Manager.exe
[post_tools_install.bat] 1 file(s) copied.
Installed Android SDK Tools, revision 15
Downloading SDK Platform Android 4.0, API 14, revision 1
Installing SDK Platform Android 4.0, API 14, revision 1
Installed SDK Platform Android 4.0, API14, revision 1
Downloading SDK Platform Android 3.2, API 13, revision 1
Downloading SDK Platform Android 1.5, API 3, revision 4 (62%, 1517 KiB/s, 13 seconds left)
Close

Figure 1-33. Android SDK Manager Log window

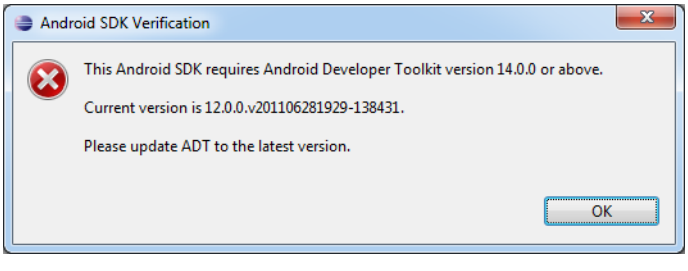

Android SDK Verification
This Android SDK requires Android Developer Toolkit version 14.0.0 or above.
Current version is 12.0.0.v201106281929-138431.
Please update ADT to the latest version.
OK

Figure 1-34. Android SDK version incorrect

In Eclipse, select Help and then select Check for Updates. Wait for the progress dialog to finish and the Android Eclipse updates will be shown. Click Next twice, and accept the licensing terms. Then click Finish to start the download and update process. A warning message about unsigned content may appear. Click OK to accept the warning (only do so if you are updating via Eclipse). Restart Eclipse once the update has completed (a message to do so will appear).

Further information on troubleshooting the SDK Manager and Android Eclipse plug-in is available on the Android Developers website.

See Also

Recipe 1.5; Installing the SDK; Adding SDK Components; ADT Plugin for Eclipse

1.13 Taking a Screenshot from the Emulator/Android Device

Rachee Singh

Problem

You want to take a screenshot of an application running on an Android device.

Solution

Use the Device Screen Capture feature of the Dalvik Debug Monitor Server (DDMS) view in Eclipse.

Discussion

To use the Device Screen Capture feature follow these steps:

1. Run the application in Eclipse and go to the DDMS view (Window menu→Open Perspective→Other→DDMS) or Window menu→Show View→Other→Android→Devices; the former is shown in Figure 1-36).

   Note that the line that reads "Resource...does not exist" appears in Figure 1-35 only because another Eclipse project has been closed, and does not affect the steps listed here.

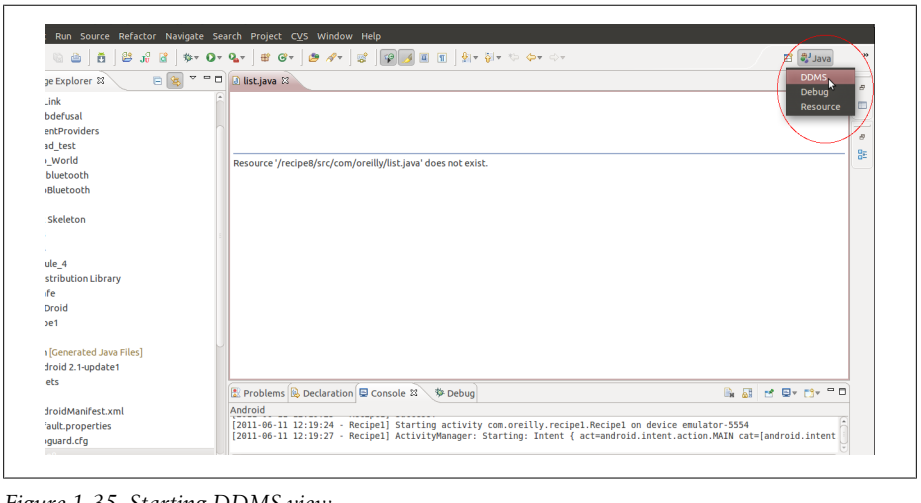

Run Source Refactor Navigate Search Project CVS Window Help
Java
DDMS
Debug
Resource
list.java
Resource '/recipe8/src/com/oreilly/list.java' does not exist.
Problems Declaration Console Debug
Android
[2011-06-11 12:19:24 - Recipe1] Starting activity com.oreilly.recipe1.Recipe1 on device emulator-5554
[2011-06-11 12:19:27 - Recipe1] ActivityManager: Starting: Intent { act=android.intent.action.MAIN cat=[android.intent

Figure 1-35. Starting DDMS view

2. In the DDMS view, select the device or emulator whose screen you want to capture.
3. In the DDMS view, click the Screen Capture icon. See Figure 1-36.

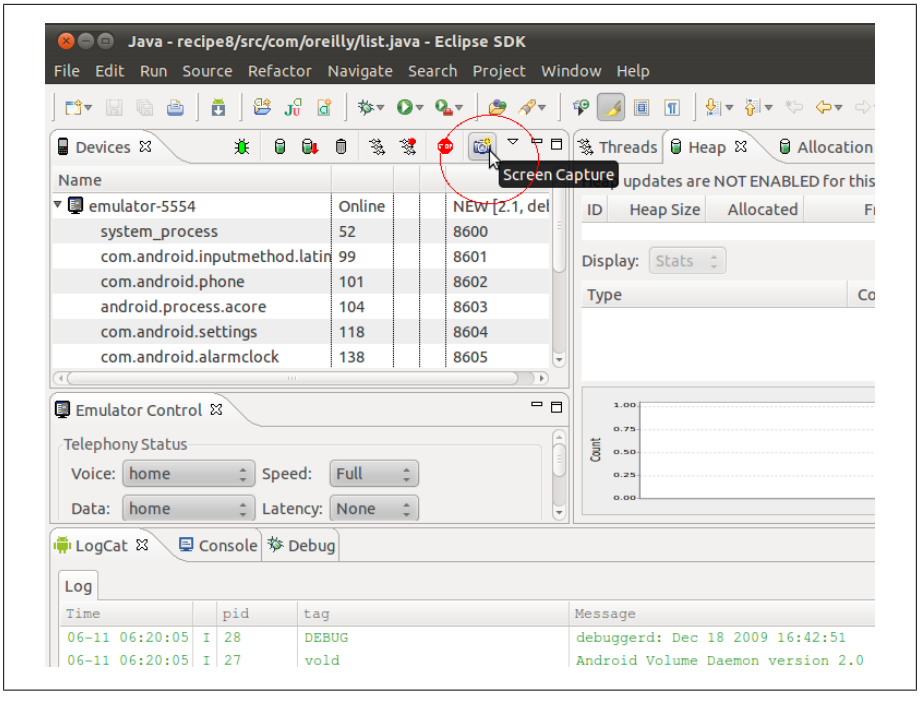


Figure 1-36. Device screen capture

4. A window showing the current screen of the emulator/Android device will pop up. It should look like Figure 1-37. You can save the screenshot and use it to describe the app!

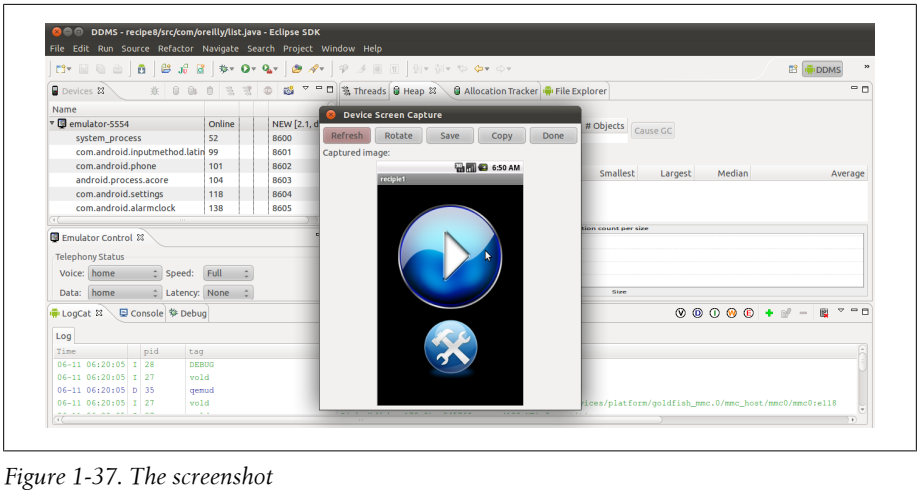


Figure 1-37. The screenshot

See Also

Some distributions provide alternative ways of taking screenshots. CyanogenMod 7.x provides a screenshot in the menu you get when you long-press the power button. Some HTC tablets with pen support offer screen grabs in the Pen menu. Ice Cream Sandwich (Android 4.0) provides a built-in mechanism for taking screenshots on real devices: just press the Volume Down control at the same time as the Power button, and the image will be saved to your device and can be viewed in the Gallery application.

1.14 Program: A Simple CountDownTimer Example

Wagied Davids

Problem

You want a simple countdown timer, a program that will count down a given number of seconds until it reaches zero.

Solution

Android comes with a built-in class for constructing `CountDownTimer`s. It's easy to use, it's efficient, and it works (that goes without saying!).

Discussion

The steps to provide a countdown timer are as follows:

1. Create a subclass of `CountDownTimer`. This class's constructor takes two arguments, `CountDownTimer(long millisInFuture, long countDownInterval)`. The first is the number of milliseconds from now when the interval should be done; at this point the subclass's `onFinish()` method will be called. The second is the frequency in milliseconds of how often you want to get notified that the timer is still running, typically to update a progress monitor or otherwise communicate with the user. Your subclass's `onTick()` method will be called with each passage of this many milliseconds.
2. Override the `onTick()` and `onFinish()` methods.
3. Instantiate a new instance in your Android Activity.
4. Call the start() method on the new instance created!

The example Countdown Timer program consists of an XML Layout (shown in Example 1-4) and some Java code (shown in Example 1-5). When run, it should look something like Figure 1-38, though the times will probably be different.

Example 1-4. main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <Button
        android:id="@+id/button"
        android:text="Start"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content" />
    <TableLayout
        android:padding="10dip"
        android:layout_gravity="center"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content">
        <TableRow>
            <TextView
                android:id="@+id/timer"
                android:text="Time: "
                android:paddingRight="10dip"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content" />

            <TextView
                android:id="@+id/timeElapsed"
                android:text="Time elapsed: "
                android:paddingRight="10dip"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content" />
        </TableRow>
    </TableLayout>
</LinearLayout>
```

Example 1-5. Main.java

```
package com.examples;

import android.app.Activity;
import android.os.Bundle;
import android.os.CountDownTimer;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;

public class Main extends Activity implements OnClickListener
    {
        private MalibuCountDownTimer countDownTimer;
        private long timeElapsed;
        private boolean timerHasStarted = false;
        private Button startB;
        private TextView text;
        private TextView timeElapsedView;
```

```
private final long startTime = 50 * 1000;
private final long interval = 1 * 1000;

/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        startB = (Button) this.findViewById(R.id.button);
        startB.setOnClickListener(this);

        text = (TextView) this.findViewById(R.id.timer);
        timeElapsedView = (TextView) this.findViewById(R.id.timeElapsed);
        countDownTimer = new MalibuCountDownTimer(startTime, interval);
        text.setText(text.getText() + String.valueOf(startTime));
    }

@Override
public void onClick(View v)
    {
        if (!timerHasStarted)
            {
                countDownTimer.start();
                timerHasStarted = true;
                startB.setText("Start");
            }
        else
            {

                countDownTimer.cancel();
                timerHasStarted = false;
                startB.setText("RESET");
            }
    }

// CountDownTimer class
public class MalibuCountDownTimer extends CountDownTimer
    {

        public MalibuCountDownTimer(long startTime, long interval)
            {
                super(startTime, interval);
            }

        @Override
        public void onFinish()
            {
                text.setText("Time's up!");
                timeElapsedView.setText("Time Elapsed: " +
                    String.valueOf(startTime));
            }

        @Override
```

```
            public void onTick(long millisUntilFinished)
                {
                    text.setText("Time remain:" + millisUntilFinished);
                    timeElapsed = startTime - millisUntilFinished;
                    timeElapsedView.setText("Time Elapsed: " +
                        String.valueOf(timeElapsed));
                }
        }
}
```

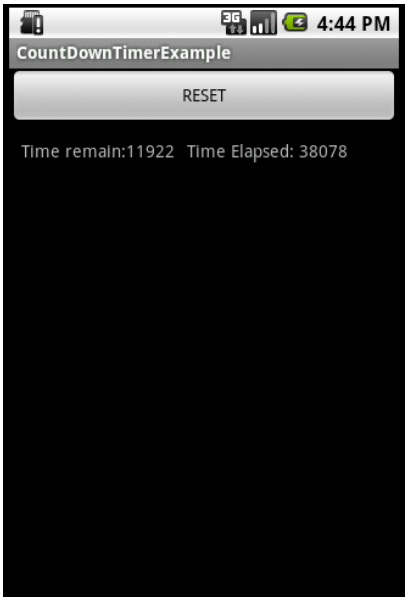


Figure 1-38. Timer reset

Source Download URL

The source code for this example is in the Android Cookbook repository at *http://github.com/AndroidCook/Android-Cookbook-Examples*, in the subdirectory CountDownTimerExample (see "Getting and Using the Code Examples" on page xvi).

1.15 Program: Tipster, a Tip Calculator for the Android OS

Sunit Katkar

Problem

When you go with friends to a restaurant and wish to divide the check and tip, you can get into a lot of manual calculations and disagreements. Instead, you want to use an

app that lets you simply add the tip percentage to the total and divide by the number of diners. Tipster is an implementation of this in Android, to show a complete application.

Solution

This is a simple exercise that uses the basic GUI elements in Android and then pieces them together with some simple calculations and some event-driven UI code to tie it all together. We will use the following GUI components:

`TableLayout`
: This provides a good control over screen layout. This layout allows you to use the HTML `Table` tag paradigm to lay out widgets.

`TableRow`
: This defines a row in the `TableLayout`. It's like the HTML `TR` and `TD` tags combined.

`TextView`
: This `View` provides a label for displaying static text on the screen.

`EditText`
: This `View` provides a text field for entering values.

`RadioGroup`
: This groups together radio buttons.

`RadioButton`
: This provides a radio button.

`Button`
: This is the regular button.

`View`
: We will use a `View` to create a visual separator with certain height and color attributes.

Discussion

Android uses XML files for the layout of widgets. In our example project, the Android plug-in for Eclipse generates a *main.xml* file for the layout. This file has the XML-based definitions of the different widgets and their containers.

There is a *strings.xml* file which has all the string resources used in the application. A default *icon.png* file is provided for the application icon.

Then there is the *R.java* file which is automatically generated (and updated when any changes are made to *main.xml*). This file has the constants defined for each layout and widget. Do not edit this file manually; the plug-in does it for you when you make any changes to your XML files.

In our example we have *Tipster.java* as the main Java file for the `Activity`.

Recipe 1.4 as well as various Google tutorials highlight how to use the plug-in. Using the Eclipse plug-in, create an Android project named Tipster. The end result will be a project layout that looks like the one shown in Figure 1-39.

Creating the layout and placing the widgets

The end goal is to create a layout similar to the one shown in Figure 1-39.

For this screen layout we will use the following layouts and widgets:

`TableLayout`
: Provides good control over screen layout. This layout allows you to use the HTML `Table` tag paradigm to lay out widgets.

`TableRow`
: This defines a row in the `TableLayout`. It's like the HTML `TR` and `TD` tags combined.

`TextView`
: This `View` provides a label for displaying static text on the screen.

`EditText`
: This `View` provides a text field for entering values.

`RadioGroup`
: This groups together radio buttons.

`RadioButton`
: This provides a radio button.

`Button`
: This is the regular button.

`View`
: We will use a View to create a visual separator with certain height and color attributes.

Familiarize yourself with these widgets as you will be using these quite a lot in applications you build. When you go to the Javadocs for layout and widget, look up the XML attributes. This will help you correlate the usage in the *main.xml* layout file and the Java code (*Tipster.java* and *R.java*) where these are accessed.

Also available is a visual layout editor in the Eclipse ADT, as well as a standalone UI tool called DroidDraw, both of which let you create a layout by dragging and dropping widgets from a palette, like any form designer tool. However, I recommend that you create the layout by hand in XML, at least in your initial stages of learning Android. Later on, as you learn all the nuances of the XML layout API, you can delegate the task to such tools.

The layout file, *main.xml*, has the layout information (see Example 1-6). A `TableRow` widget creates a single row inside the `TableLayout`. So you use as many `TableRow`s as the number of rows you want. In this tutorial we will use eight `TableRow`s—five for the

widgets up to the visual separator below the buttons, and three for the results area below the buttons and separator.

Example 1-6. /res/layout/main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Using table layout to have HTML table like control over layout -->
<TableLayout
        android:id="@+id/TableLayout01"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:stretchColumns="1"
        xmlns:android="http://schemas.android.com/apk/res/android">
    <!-- Row 1: Text label placed in column zero,
         text field placed in column two and allowed to
         span two columns. So a total of 4 columns in this row -->
        <TableRow>
        <TextView
                android:id="@+id/txtLbl1"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_column="0"
                android:text="@string/textLbl1"/>
        <EditText❶
                android:id="@+id/txtAmount"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:numeric="decimal"
                android:layout_column="2"
                android:layout_span="2"
                />
        </TableRow>
    <!-- Row 2: Text label placed in column zero,
         text field placed in column two and allowed to
         span two columns. So a total of 4 columns in this row -->
        <TableRow>
        <TextView
                android:id="@+id/txtLbl2"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_column="0"
                android:text="@string/textLbl2"/>
        <EditText
                android:id="@+id/txtPeople"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:numeric="integer"
                android:layout_column="2"
                android:layout_span="3"/>
        </TableRow>
   <!-- Row 3: This has just one text label placed in column zero  -->
        <TableRow>
        <TextView
                android:id="@+id/txtLbl3"
                android:layout_width="wrap_content"
```

```
            android:layout_height="wrap_content"
            android:text="@string/textLbl3"/>
    </TableRow>
<!-- Row 4: RadioGroup for RadioButtons placed at column zero
    with column span of three, thus creating one radio button
    per cell of the table row. Last cell number 4 has the
    textfield to enter a custom tip percentage -->
    <TableRow>
    <RadioGroup
            android:id="@+id/RadioGroupTips"
            android:orientation="horizontal"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_column="0"
            android:layout_span="3"
            android:checkedButton="@+id/radioFifteen">
            <RadioButton android:id="@+id/radioFifteen"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:text="@string/rdoTxt15"
                    android:textSize="15sp" />
            <RadioButton android:id="@+id/radioTwenty"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:text="@string/rdoTxt20"
                    android:textSize="15sp" />
            <RadioButton android:id="@+id/radioOther"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:text="@string/rdoTxtOther"
                    android:textSize="15sp" />
    </RadioGroup>
            <EditText
                    android:id="@+id/txtTipOther"
                    android:layout_width="fill_parent"
                    android:layout_height="wrap_content"
                    android:numeric="decimal"/>
    </TableRow>
<!--  Row for the Calculate and Rest buttons. The Calculate button
     is placed at column two, and Reset at column three -->
    <TableRow>
    <Button
            android:id="@+id/btnReset"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_column="2"
            android:text="@string/btnReset"/>
    <Button
            android:id="@+id/btnCalculate"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_column="3"
            android:text="@string/btnCalculate"/>
    </TableRow>
```

```
<!-- TableLayout allows any other views to be inserted between
     the TableRow elements. So insert a blank view to create a
     line separator. This separator view is used to separate
     the area below the buttons which will display the
     calculation results -->
    <View
            android:layout_height="2px"
            android:background="#DDFFDD"
            android:layout_marginTop="5dip"
            android:layout_marginBottom="5dip"/>

<!-- Again table row is used to place the result textviews
     at column zero and the result in textviews at column two -->
    <TableRow android:paddingBottom="10dip" android:paddingTop="5dip">
    <TextView
            android:id="@+id/txtLbl4"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_column="0"
            android:text="@string/textLbl4"/>
    <TextView
            android:id="@+id/txtTipAmount"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_column="2"
            android:layout_span="2"/>
    </TableRow>

    <TableRow android:paddingBottom="10dip" android:paddingTop="5dip">
    <TextView
            android:id="@+id/txtLbl5"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_column="0"
            android:text="@string/textLbl5"/>
    <TextView
            android:id="@+id/txtTotalToPay"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_column="2"
            android:layout_span="2"/>
    </TableRow>

    <TableRow android:paddingBottom="10dip" android:paddingTop="5dip">
    <TextView
            android:id="@+id/txtLbl6"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_column="0"
            android:text="@string/textLbl6"/>
    <TextView
            android:id="@+id/txtTipPerPerson"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_column="2"
```

```
                android:layout_span="2"/>
        </TableRow>
    <!--  End of all rows and widgets -->
</TableLayout>
```

TableLayout and TableRow

After examining *main.xml*, you can gather that the `TableLayout` and `TableRow` are straightforward to use. You create the `TableLayout` once, then insert a `TableRow`. Now you are free to insert any other widgets, such as `TextView`, `EditView`, and so on, inside this `TableRow`.

Do look at the attributes, especially `android:stretchColumns`, `android:layout_column`, and `android:layout_span`, which allow you to place widgets the same way you would use a regular HTML table. I recommend that you follow the links to these attributes and read up on how they work for a `TableLayout`.

Controlling input values

Controlling input values: Look at the `EditText` widget in the *main.xml* file at ❶. This is the first text field for entering the "Total Amount" of the check. We want only numbers here. We can accept decimal numbers because real restaurant checks can be for dollars and cents, and not just dollars. So we use the `android:numeric` attribute with a value of `decimal`. This will allow whole values like 10 and decimal values like 10.12, but will prevent any other type of entry.

This is a simple and concise way to control input values, thus saving us the trouble of writing validation code in the *Tipster.java* file, and ensuring that the user does not enter erroneous values. This XML-based constraints feature of Android is quite powerful and useful. You should explore all possible attributes that go with a particular widget to extract maximum benefits from this XML shorthand way of setting constraints. In a future release, unless I have missed it completely in this release, I hope that Android allows for entering ranges for the `android:numeric` attribute so that we can define what range of numbers we wish to accept.

Since ranges are not currently available (to the best of my knowledge), you will see later on that we do have to check for certain values like zero or empty values to ensure that our tip calculation arithmetic does not fail.

Examining Tipster.java

Now we will look at the *Tipster.java* file which controls our application. This is the main class that does the layout, the event handling, and the application logic.

The Android Eclipse plug-in creates the *Tipster.java* file in our project with the default code shown in Example 1-7.

Example 1-7. Code snippet 1 of /src/com/examples/tipcalc/Tipster.java

```
package com.examples.tipcalc;

import android.app.Activity;

public class Tipster extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

The `Tipster` class extends the `android.app.Activity` class. An activity is a single, focused thing that the user can do. The `Activity` class takes care of creating the window and then laying out the UI. You have to call the `setContentView(View view)` method to put your UI in the `Activity`. So think of `Activity` as an outer frame that is empty, and that you populate with your UI.

Now look at the snippet of the *Tipster.java* class shown in Example 1-8. First we define the widgets as class members. Look at ❶ through ❷ in particular for reference.

Then we use the `findViewById(int id)` method to locate the widgets. The ID of each widget, defined in your *main.xml* file, is automatically defined in the *R.java* file when you clean and build the project in Eclipse. (If you have set up Eclipse to build automatically, the *R.java* file is instantaneously updated when you update *main.xml*.)

Each widget is derived from the `View` class, and provides special GUI features. So a `TextView` provides a way to put labels on the UI, while the `EditText` provides a text field. Look at ❸ through ❻ in Example 1-8. You can see how `findViewById()` is used to locate the widgets.

Example 1-8. Code snippet 2 of /src/com/examples/tipcalc/Tipster.java

```
public class Tipster extends Activity {
    // Widgets in the application
    private EditText txtAmount;❶
    private EditText txtPeople;
    private EditText txtTipOther;
    private RadioGroup rdoGroupTips;
    private Button btnCalculate;
    private Button btnReset;

    private TextView txtTipAmount;
    private TextView txtTotalToPay;
    private TextView txtTipPerPerson;❷

    // For the id of radio button selected
    private int radioCheckedId = -1;

    /** Called when the activity is first created. */
```

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    // Access the various widgets by their id in R.java
    txtAmount = (EditText) findViewById(R.id.txtAmount);❸
    //On app load, the cursor should be in the Amount field
    txtAmount.requestFocus();❹

    txtPeople = (EditText) findViewById(R.id.txtPeople);
    txtTipOther = (EditText) findViewById(R.id.txtTipOther);

    rdoGroupTips = (RadioGroup) findViewById(R.id.RadioGroupTips);

    btnCalculate = (Button) findViewById(R.id.btnCalculate);
    //On app load, the Calculate button is disabled
    btnCalculate.setEnabled(false);❺

    btnReset = (Button) findViewById(R.id.btnReset);

    txtTipAmount = (TextView) findViewById(R.id.txtTipAmount);
    txtTotalToPay = (TextView) findViewById(R.id.txtTotalToPay);
    txtTipPerPerson = (TextView) findViewById(R.id.txtTipPerPerson);❻

    // On app load, disable the Other Tip Percentage text field
    txtTipOther.setEnabled(false);❼
```

Addressing ease of use or usability concerns

Our application must try to be as usable as any other established application or web page. In short, adding usability features will result in a good user experience. To address these concerns look at Example 1-8 again.

Look at ❹ where we use the `requestFocus()` method of the `View` class. Since the `EditText` widget is derived from the `View` class, this method is applicable to it. This is done so that when our application loads the Total Amount text field will receive focus and the cursor will be placed in it. This is similar to popular web application login screens where the cursor is present in the username text field.

Now look at ❺ where the Calculate button is disabled by calling the `setEnabled(boolean enabled)` method on the `Button` widget. This is done so that the user cannot click on it before entering values in the required fields. If we allowed the user to click Calculate without entering values in the Total Amount and No. of People fields, we would have to write validation code to catch these conditions. This would entail showing an alert pop up warning the user about the empty values. This adds unnecessary code and user interaction. When the user sees the Calculate button disabled, it's quite obvious that unless all values are entered, the tip cannot be calculated.

Look at ❼ in Example 1-8. Here the Other Tip Percentage text field is disabled. This is done because the "15% tip" radio button is selected by default when the application

loads. This default selection on application load is done via the *main.xml* file. Look at the line of *main.xml* where the following statement selects the "15% tip" radio button:

```
android:checkedButton="@+id/radioFifteen"
```

The `RadioGroup` attribute `android:checkedButton` allows you to select one of the `RadioButton` widgets in the group by default.

Most users who have used popular applications on the desktop as well as the Web are familiar with the "disabled widgets enabled on certain conditions" paradigm. Adding such small conveniences always makes an application more usable and the user experience richer.

Processing UI events

Like popular Windows, Java Swing, Flex, and other UI frameworks, Android also provides an event model which allows you to listen to certain events in the UI caused by user interaction. Let's see how we can use the Android event model in our application.

First let's focus on the radio buttons in the UI. We want to know which radio button the user selected, as this will allow us to determine the tip percentage in our calculations. To "listen" to radio buttons, we use the static interface `OnCheckedChangeListener()`. This will notify us when the selection state of a radio button changes.

In our application, we want to enable the Other Tip Percentage text field only when the Other radio button is selected. When the "15% tip" and "20% tip" buttons are selected we want to disable this text field. Besides this, we want to add some more logic for the sake of usability. As we discussed before, we should not enable the Calculate button until all the required fields have valid values. In terms of the three radio buttons, we want to ensure that the Calculate button gets enabled for the following two conditions:

- The Other radio button is selected and the Other Tip Percentage text field has valid values.
- The "15% tip" or "20% tip" radio button is selected and the Total Amount and No. of People text fields have valid values

Look at Example 1-9, which deals with the radio buttons. The source code comments are quite self-explanatory.

Example 1-9. Code snippet 3 of /src/com/examples/tipcalc/Tipster.java

```
/*
 * Attach an OnCheckedChangeListener to the
 * radio group to monitor radio buttons selected by user
 */
rdoGroupTips.setOnCheckedChangeListener(new OnCheckedChangeListener() {

@Override
public void onCheckedChanged(RadioGroup group, int checkedId) {
   // Enable/disable Other Tip Percentage field
```

```
    if (checkedId == R.id.radioFifteen
            || checkedId == R.id.radioTwenty) {
        txtTipOther.setEnabled(false);
        /*
         * Enable the calculate button if Total Amount and No. of
         * People fields have valid values.
         */
        btnCalculate.setEnabled(txtAmount.getText().length() > 0
                && txtPeople.getText().length() > 0);
    }
    if (checkedId == R.id.radioOther) {
       // enable the Other Tip Percentage field
       txtTipOther.setEnabled(true);
       // set the focus to this field
       txtTipOther.requestFocus();
       /*
        * Enable the calculate button if Total Amount and No. of
        * People fields have valid values. Also ensure that user
        * has entered an Other Tip Percentage value before enabling
        * the Calculate button.
        */
       btnCalculate.setEnabled(txtAmount.getText().length() > 0
               && txtPeople.getText().length() > 0
               && txtTipOther.getText().length() > 0);
    }
    // To determine the tip percentage choice made by user
    radioCheckedId = checkedId;
   }
});
```

Monitoring key activity in text fields

As I mentioned earlier, the Calculate button must not be enabled unless the text fields have valid values. So we have to ensure that the Calculate button will be enabled only if the Total Amount, No. of People, and Other Tip Percentage text fields have valid values. The Other Tip Percentage text field is enabled only if the Other Tip Percentage radio button is selected.

We do not have to worry about the type of values, that is, whether the user entered negative values or letters because the `android:numeric` attribute has been defined for the text fields, thus limiting the types of values that the user can enter. We have to just ensure that the values are present.

So we use the static interface `OnKeyListener()`. This will notify us when a key is pressed. The notification reaches us before the actual key pressed is sent to the `EditText` widget.

Look at the code in Examples 1-10 and 1-11 which deal with key events in the text fields. As in Example 1-9, the source code comments are quite self-explanatory.

Example 1-10. Code snippet 4 of /src/com/examples/tipcalc/Tipster.java

```
/*
 * Attach a KeyListener to the Tip Amount, No. of People and Other Tip
 * Percentage text fields
 */
txtAmount.setOnKeyListener(mKeyListener);
txtPeople.setOnKeyListener(mKeyListener);
txtTipOther.setOnKeyListener(mKeyListener);
```

Notice that we create just one listener instead of creating anonymous/inner listeners for each text field. I am not sure if my style is better or recommended, but I always write in this style if the listeners are going to perform some common actions. Here the common concern for all the text fields is that they should not be empty, and only when they have values should the Calculate button be enabled.

Example 1-11. Code snippet 5 from KeyListener.java

```
/*
 * KeyListener for the Total Amount, No of People and Other Tip Percentage
 * text fields. We need to apply this key listener to check for the following
 * conditions:
 *
 * 1) If the user selects Other Tip Percentage, then the Other Tip Percentage text field
 * should have a valid tip percentage entered by the user. Enable the
 * Calculate button only when the user enters a valid value.
 *
 * 2) If the user does not enter values in the Total Amount and No. of People fields,
 * we cannot perform the calculations. Hence we enable the Calculate button
 * only when the user enters valid values.
 */
private OnKeyListener mKeyListener = new OnKeyListener() {
    @Override
    public boolean onKey(View v, int keyCode, KeyEvent event) {

    switch (v.getId()) {❶
    case R.id.txtAmount:❷
    case R.id.txtPeople:❸
        btnCalculate.setEnabled(txtAmount.getText().length() > 0
                && txtPeople.getText().length() > 0);
        break;
    case R.id.txtTipOther:❹
        btnCalculate.setEnabled(txtAmount.getText().length() > 0
                && txtPeople.getText().length() > 0
                && txtTipOther.getText().length() > 0);
        break;
    }
    return false;
    }

};
```

At ❶ in Example 1-11, we examine the ID of the `View`. Remember that each widget has a unique ID as we define it in the *main.xml* file. These values are then defined in the generated *R.java* class.

At ❷ and ❸, if the key event occurred in the Total Amount or No. of People fields, we check for the value entered in the field. We are ensuring that the user has not left both fields blank.

At ❹ we check if the user has selected the Other radio button, and then we ensure that the Other text field is not empty. We also check once again if the Total Amount and No. of People fields are empty.

So the purpose of our `KeyListener` is now clear: ensure that all text fields are not empty and only then enable the Calculate button.

Listening to button clicks

Now we will look at the Calculate and Reset buttons. When the user clicks these buttons, we use the static interface `OnClickListener()` which will let us know when a button is clicked.

As we did with the text fields, we create just one listener and within it we detect which button was clicked. Depending on the button that was clicked, the `calculate()` or `reset()` method is called.

Example 1-12 shows how the click listener is added to the buttons.

Example 1-12. Code snippet 6 of /src/com/examples/tipcalc/Tipster.java

```
/* Attach listener to the Calculate and Reset buttons */
btnCalculate.setOnClickListener(mClickListener);
btnReset.setOnClickListener(mClickListener);
```

Example 1-13 shows how to detect which button is clicked by checking for the ID of the `View` that receives the click event.

Example 1-13. Code snippet 7 of /src/com/examples/tipcalc/Tipster.java

```
/**
 * ClickListener for the Calculate and Reset buttons.
 * Depending on the button clicked, the corresponding
 * method is called.
 */
private OnClickListener mClickListener = new OnClickListener() {

    @Override
    public void onClick(View v) {
        if (v.getId() == R.id.btnCalculate) {
            calculate();
        } else {
            reset();
        }
```

```
    }
};
```

Resetting the application

When the user clicks the Reset button, the text fields should be cleared, the default "15% tip" radio button should be selected, and any results calculated should be cleared.

Example 1-14 shows the `reset()` method.

Example 1-14. Code snippet 8 of /src/com/examples/tipcalc/Tipster.java

```
/**
 * Resets the results text views at the bottom of the screen as well as
 * resets the text fields and radio buttons.
 */
private void reset() {
    txtTipAmount.setText("");
    txtTotalToPay.setText("");
    txtTipPerPerson.setText("");
    txtAmount.setText("");
    txtPeople.setText("");
    txtTipOther.setText("");
    rdoGroupTips.clearCheck();
    rdoGroupTips.check(R.id.radioFifteen);
    // set focus on the first field
    txtAmount.requestFocus();
}
```

Validating the input to calculate the tip

As I said before, we are limiting what type of values the user can enter in the text fields. However, the user could still enter a value of zero in the Total Amount, No. of People, and Other Tip Percentage text fields, thus causing error conditions like divide by zero in our tip calculations.

If the user enters zero we must show an alert pop up asking the user to enter non-zero values. We handle this with a method called `showErrorAlert(String errorMessage, final int fieldId)`, but we will discuss this in more detail later.

First, look at Example 1-15 which shows the `calculate()` method. Notice how the values entered by the user are parsed as double values.

Now notice ❶ and ❷ where we check for zero values. If the user enters zero, we show an alert pop up to warn the user. Next, look at ❸, where the Other Tip Percentage text field is enabled because the user selected the Other radio button. Here, too, we must check for the tip percentage being zero.

When the application loads, the "15% tip" radio button is selected by default. If the user changes the selection, we assign the ID of the selected radio button to the member variable `radioCheckedId`, as we saw in Example 1-9, in `OnCheckedChangeListener`.

But if the user accepts the default selection, the `radioCheckedId` will have the default value of `-1`. In short, we will never know which radio button was selected. Of course, we know which one is selected by default and could have coded the logic slightly differently, to assume 15% if `radioCheckedId` has the value `-1`. But if you refer to the API, you will see that we can call the method `getCheckedRadioButtonId()` on the `RadioGroup` and not on individual radio buttons. This is because `OnCheckedChangeListener` readily provides us with the ID of the radio button selected.

Showing the results

Calculating the tip is simple. If there are no validation errors, the boolean flag `isError` will be `false`. Look at ❹ through ❺ in Example 1-15 for the simple tip calculations. Next, the calculated values are set to the `TextView` widgets from ❻ to ❼.

Example 1-15. Code snippet 9 of /src/com/examples/tipcalc/Tipster.java

```
/**
 * Calculate the tip as per data entered by the user.
 */
private void calculate() {
    Double billAmount = Double.parseDouble(
        txtAmount.getText().toString());
    Double totalPeople = Double.parseDouble(
        txtPeople.getText().toString());
    Double percentage = null;
    boolean isError = false;
    if (billAmount < 1.0) {❶
        showErrorAlert("Enter a valid Total Amount.",
            txtAmount.getId());
        isError = true;
    }

    if (totalPeople < 1.0) {❷
        showErrorAlert("Enter a valid value for No. of People.",
            txtPeople.getId());
        isError = true;
    }

    /*
     * If the user never changes his radio selection, then it means
     * the default selection of 15% is in effect. But it's
     * safer to verify
     */
    if (radioCheckedId == -1) {
        radioCheckedId = rdoGroupTips.getCheckedRadioButtonId();
    }
    if (radioCheckedId == R.id.radioFifteen) {
        percentage = 15.00;
    } else if (radioCheckedId == R.id.radioTwenty) {
        percentage = 20.00;
    } else if (radioCheckedId == R.id.radioOther) {
        percentage = Double.parseDouble(
            txtTipOther.getText().toString());
```

```
        if (percentage < 1.0) {❸
            showErrorAlert("Enter a valid Tip percentage",
                txtTipOther.getId());
            isError = true;
        }
    }

    /*
     * If all fields are populated with valid values, then proceed to
     * calculate the tips
     */
    if (!isError) {
        Double tipAmount = ((billAmount * percentage) / 100);❹
        Double totalToPay = billAmount + tipAmount;
        Double perPersonPays = totalToPay / totalPeople;❺

        txtTipAmount.setText(tipAmount.toString());❻
        txtTotalToPay.setText(totalToPay.toString());
        txtTipPerPerson.setText(perPersonPays.toString());❼
    }
}
```

Showing the alerts

Showing the alerts Android provides the `AlertDialog` class to show alert pop ups. This lets us show a dialog with up to three buttons and a message.

Example 1-16 shows the `showErrorAlert` method which uses this `AlertDialog` to show the error messages. Notice that we pass two arguments to this method: `String errorMessage` and `int fieldId`. The first argument is the error message we want to show to the user. The `fieldId` is the ID of the field which caused the error condition. After the user dismisses the alert dialog, this `fieldId` will allow us to request the focus on that field, so the user knows which field has the error.

Example 1-16. Code snippet 10 of /src/com/examples/tipcalc/Tipster.java

```
/**
 * Shows the error message in an alert dialog
 *
 * @param errorMessage
 *            String for the error message to show
 * @param fieldId
 *            the Id of the field which caused the error.
 *            This is required so that the focus can be
 *            set on that field once the dialog is
 *            dismissed.
 */
private void showErrorAlert(String errorMessage,
    final int fieldId) {
    new AlertDialog.Builder(this).setTitle("Error")
    .setMessage(errorMessage).setNeutralButton("Close",
            new DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface dialog,
```

```
                    int which) {
                findViewById(fieldId).requestFocus();
            }
        }).show();
}
```

When all this is put together, it should look like Figure 1-39.

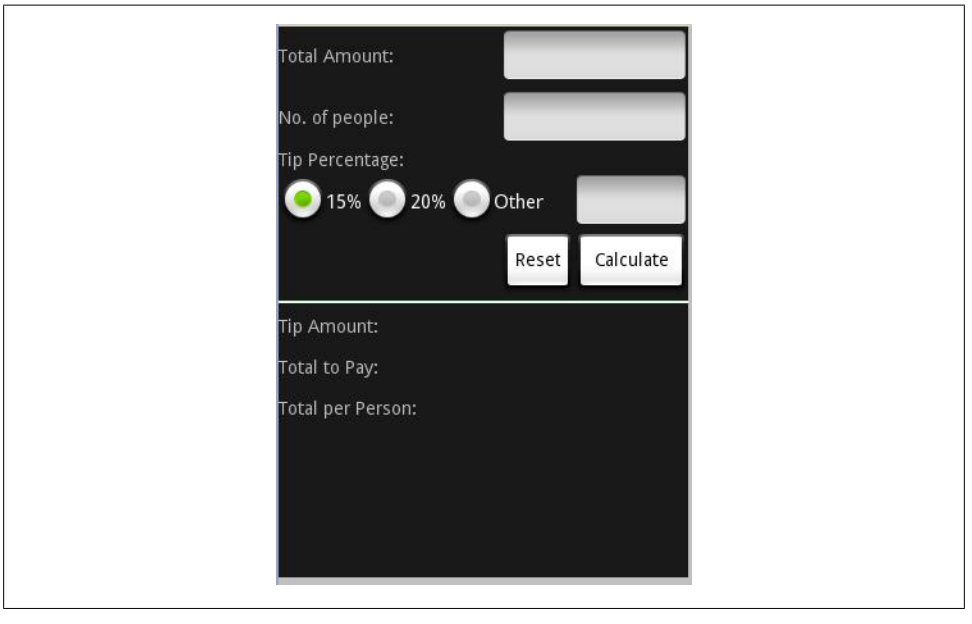


Figure 1-39. Tipster in action

Conclusion

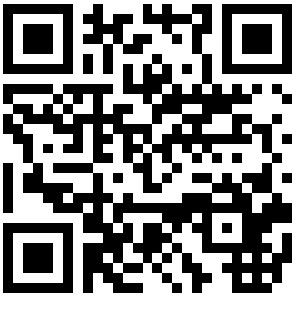
Developing for the Android OS is not too different from developing for any other UI toolkit, including Microsoft Windows, X Windows, Java Swing, or Adobe Flex. Of course Android has its differences and, overall, a very good design. The XML layout paradigm is quite cool and useful for building complex UIs using simple XML. In addition, the event handling model is simple, feature-rich, and intuitive to use in code.

Source Download URL

You can download the source code for this example from *http://www.vidyut.com/sunit/android/tipster.zip*.

Binary Download URL

You can download the executable code for this example from *http://www.vidyut.com/sunit/android/tipster.zip*.



O'Reilly Ebooks—Your bookshelf on your devices!

When you buy an ebook through oreilly.com you get lifetime access to the book, and whenever possible we provide it to you in five, DRM-free file formats—PDF, .epub, Kindle-compatible .mobi, Android .apk, and DAISY—that you can use on the devices of your choice. Our ebook files are fully searchable, and you can cut-and-paste and print them. We also alert you when we've updated the files with corrections and additions.

Learn more at ebooks.oreilly.com

You can also purchase O'Reilly ebooks through the iBookstore, the Android Marketplace, and Amazon.com.

O'REILLY®

Spreading the knowledge of innovators

oreilly.com